Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA
DE SEGOVIA

Grado en Ingeniería Informática
de Servicios y Aplicaciones

Agentes inteligentes en videojuegos: Bloons TD 6

Alumno: Gustavo Cortés Jiménez

Tutores: Aníbal Bregón Bregón
Jorge Silvestre Vilches

Fecha: 13 de julio de 2023

Agentes inteligentes en videojuegos: Bloons TD 6

Gustavo Cortés Jiménez

13 de julio de 2023

A mis padres y hermano,
por siempre estar a mi lado.

"El 90 % de lo que se considera "imposible"
es, de hecho, posible.
El 10 % restante será posible
con el paso del tiempo y la tecnología".
– Hideo Kojima

"Los programadores del mañana
son los magos del futuro.
Parecerá que tienen poderes mágicos
en comparación con los demás".
– Gabe Newell

Resumen

La Inteligencia Artificial es una herramienta vital para los videojuegos desde los inicios de este medio de entretenimiento en la década de los 70. Pero tanto la Inteligencia Artificial como los videojuegos han sufrido una gran evolución desde entonces. Por ello, en este proyecto se desarrolla un sistema de Inteligencia Artificial por Aprendizaje por Refuerzo Profundo capaz de jugar al videojuego Bloons Tower Defense 6, de Ninja Kiwi.

Este sistema, denominado agente inteligente, es capaz de leer el estado de la partida, y a partir de él procesar cuál es la acción más adecuada para realizar a continuación. Al utilizar Aprendizaje por Refuerzo Profundo (algoritmo PPO), el agente recibe premios y castigos según las acciones que realiza, en forma de puntuación, que le permitirá aprender al buscar maximizar la puntuación total, denominada "recompensa".

BloonsTD6 es un juego complejo, con muchas variables, por lo que supone un reto para la Inteligencia Artificial. Por ello, al igual que cualquier jugador, el agente comenzará aprendiendo los elementos básicos del videojuego, y poco a poco irá adquiriendo cada vez más conocimientos sobre las mecánicas, hasta superar los modos de juego a los que se enfrente de forma autónoma.

Palabras claves: Videojuegos, Inteligencia Artificial, Bloons Tower Defense 6, PPO, Agente Inteligente.

Abstract

Artificial Intelligence has been a vital tool for video games since the beginnings of this entertainment medium in the 1970s. But both Artificial Intelligence and video games have undergone a great evolution since then. Therefore, this project develops a Deep Reinforcement Learning Artificial Intelligence system capable of playing the video game Bloons Tower Defense 6, by Ninja Kiwi.

This system, called intelligent agent, is able to read the state of the game, and from it process what is the most appropriate action to perform next. By using Deep Reinforcement Learning (PPO algorithm), the agent receives rewards and punishments according to the actions it performs, in the form of a score, which will allow it to learn by seeking to maximize the total score, called "reward".

BloonsTD6 is a complex game, with many variables, making it a challenge for Artificial Intelligence. Therefore, like any player, the agent will begin by learning the basics of the video game, and will gradually acquire more and more knowledge about the mechanics, until it overcomes the game modes that it will face autonomously.

Keywords: Video games, Artificial Intelligence, Bloons Tower Defense 6, PPO, Intelligent Agent.

Índice general

Índice de figuras

Índice de tablas

Parte I

Descripción del proyecto

Capítulo 1

Introducción

La Inteligencia Artificial es una herramienta vital para los videojuegos desde los inicios de este medio de entretenimiento en la década de los 70 [19]. Videojuegos como Space Invaders o Pacman comenzaban a utilizar Inteligencia Artifical para describir el comportamiento de sus entidades. En Space Invaders (1978), se definía la dificultad y los patrones de los enemigos según cómo interactuase el jugador. En Pacman (1980), cada fantasma presentaba una "personalidad" que describía su patrón de comportamiento. Los juegos de lucha comenzaron a utilizar Inteligencia Artificial para controlar a los rivales, como Karate Champ en 1984, y en 1988, el videojuego First Queen, del género rol de acción, fue el primero en su género en mostrar personajes controlados por la Inteligencia Artificial del ordenador. Dos años más tarde, en 1990, el videojuego de rol Dragon Quest IV introdujo un sistema táctico, en el que el jugador podía ajustar las rutinas de la Inteligencia Artificial de los personajes que no controlase durante los combates. Este concepto se siguió utilizando en otros juegos de rol, como Secret of Mana (1993).

En la década de los 90 se comenzaron a utilizar las máquinas de estados finitos en los nuevos géneros que surgieron en esta década. Por ejemplo, en los juegos de estrategia en tiempo real se utilizaba inteligencia artificical para resolver problemas de búsqueda de caminos, de toma de decisiones en tiempo real y de planificación económica. Sin embargo, estas implementaciones eran muy básicas, por ejemplo la búsqueda de caminos del juego Herzog Zwei prácticamente no funcionaba, y utilizaba máquinas de tres estados muy básicas para controlar las unidades. A medida que el campo de la Inteligencia Artificial se hizo más sofisticado, estos problemas se fueron solventando. Estas dos últimas décadas el uso de la Inteligencia Artificial en los videojuegos se ha centrado en desarrollar patrones en los personajes que los hagan parecer humanos, crear entornos "vivos" y, en general, construir una experiencia inmersiva para el jugador.

En este proyecto se desarrolla un sistema de Inteligencia Artificial capaz de jugar al videojuego Bloons Tower Defense 6, de Ninja Kiwi, un videojuego de estrategia lanzado en 2017. En este documento se estudia el entorno actual de agentes automáticos destinados a jugar juegos, comparar alternativas, y detallar el proceso de análisis, diseño, implementación y pruebas característicos del desarrollo de sistemas *software*.

1.1. Planteamiento del problema (*Problem Statement*)

El agente inteligente ideal en BloonsTD6 es aquel que fuese capaz de actuar como lo haría un humano, incluso pudiendo elegir el nivel de habilidad con el que este actuaría. Esto permitiría

utilizar al agente para evaluar nuevo contenido y encontrar nuevas estrategias, así como permitir a un usuario jugar en modo cooperativo con un agente de su mismo nivel.

Sin embargo, alcanzar este comportamiento es complejo y costoso. Hoy en día es necesario una exhuberante cantidad de tiempo, información y otros recursos para entrenar a un agente inteligente de estas características. Otro factor a tener en cuenta es el nivel de conocimiento y experiencia en el campo del aprendizaje automático, siendo básico en mi caso, por lo que es necesario un gran proceso de investigación y aprendizaje sobre este campo.

El uso de un agente inteligente para evaluar nuevas características que se fuesen a incorporar en BloonsTD6 reduciría el esfuerzo humano necesario en este tipo de tareas repetitivas, además de encontrar posibles aspectos secundarios como consecuencia de este nuevo contenido, en especial teniendo en cuenta que el juego se hace cada vez más grande y, por extensión, más complejo. Este agente tambíen podría actuar como un compañero en las partidas en modo cooperativo, ya que este modo requiere una buena conexión a Internet estable, y existen usuarios que no pueden disponer de esta conexión, provocando que su experiencia de juego sea peor.

El agente que se propone en este proyecto busca ser capaz de superar los modos de juego de BloonsTD6 de la misma forma que lo haría un jugador. Debido a la cantidad de tiempo del que se dispone, este agente será una versión básica, capaz de entender las características básicas del juego e idear y aplicar estrategias para ganar partidas sencillas. Para ello se utilizará el aprendizaje automático profundo y la simulación del teclado y ratón, periféricos con los que jugaría un usuario estándar.

1.2. Objetivos del trabajo

Este proyecto busca mostrar una implementación de Inteligencia Artificial para el videojuego BloonsTD6, al desarrollar un agente inteligente capaz de jugar a este videojuego gracias a la capacidad de aprender que le confiere el aprendizaje automático.

OBJ-01 Describir el videojuego como el entorno del agente.

OBJ-02 Implementar procedimientos de interacción del agente con el videojuego.

- **OBJ-02.1** Obtención del estado del videojuego por parte del agente.
- **OBJ-02.2** Construir mecanismos de acción del agente sobre el videojuego.

OBJ-03 Desarrollar el agente.

- **OBJ-03.1** Definir y configurar el algoritmo de aprendizaje.
- **OBJ-03.2** Entrenar al agente.

1.2.1. Restricciones

Las restricciones son limitaciones impuestas sobre las posibles decisiones que se pueden tomar en el diseño o implementacion del sistema. Pueden estar definidas por agentes externos, como es el caso de la limitación temporal del proyecto, que está definida por el número de horas que se destinan a una asignatura por crédito (12 créditos * 25 horas por crédito = 300 horas). La segunda restricción nace de la necesidad de disponer de un entorno controlado, ya que el videojuego BloonsTD6 recibe actualizaciones periódicas frecuentes (cada dos o tres meses). De esta forma,

si se modifican parámetros que se utilicen en este proyecto, no será necesario reconfigurar al agente. Se ha escogido esta versión por ser la actual, y no presentar *bugs* notables.

Finalmente, el sistema debe cumplir los Términos y Condiciones [39] de la empresa a la que pertenece BloonsTD6, Ninja Kiwi. Concretamente, el fragmento que afecta a este proyecto es el siguiente: "Fan Content must not be used to communicate, link, distribute, or promote cheats or hacks to any Ninja Kiwi games", es decir, que "el contenido creado por los fans no debe utilizarse para comunicar, enlazar, distribuir o promocionar trucos o hacks de ningún juego de Ninja Kiwi". Como el agente inteligente de este proyecto se podría utilizar para obtener una divisa dentro del juego, se podría calificar como "hack". Es por esto que no se puede hacer público el código del agente ni distribuirlo.

RE-01 La duración máxima del proyecto será de 300 horas.

RE-02 La versión del juego será la 35.2.

RE-03 Se deben cumplir los Términos y Condiciones de Ninja Kiwi.

1.3. Estructura de la memoria

En la primera parte de este documento, donde se encuentra esta sección, se describe el proyecto a alto nivel. Comienza con una explicación de los conceptos clave del proyecto (cap. 1), para a continuación describir la planificación que se ha realizado del mismo (cap. 2): qué metodología se utiliza, las estimaciones temporales y presupuestarias, cómo se han ajustado estas estimaciones con la realidad tras completar el proyecto, y el análisis de posibles riesgos.

La siguiente sección trata sobre los antecedentes del proyecto (cap. 3), donde se analizan a los posibles interesados o *stakeholders* del mismo, y se explica tanto el funcionamiento del videojuego BloonsTD6 como de las herramientas y librerías utilizadas. A continuación, se explica el contexto científico-técnico de los agentes inteligentes para proceder a estudiar y comparar otras soluciones similares al sistema *software*.

En la segunda parte de la memoria (cap. 4) se lleva a cabo el análisis de los requisitos del sistema *software*, el diseño de los componentes y sus interacciones, la implementación del agente inteligente y las pruebas de caja negra para asegurar su correcto funcionamiento; a continuación, en la tercera parte de la memoria (cap. 5), se lleva a cabo el proceso de experimentación, donde se evalúan diferentes cambios sobre el agente inteligente utilizando varias métricas sobre los resultados.

El último capítulo (cap. 6) contiene las conclusiones que han nacido del proyecto, evaluando los resultados obtenidos y evaluando el desarrollo del proyecto. Junto a esto se proponen mejoras para el futuro, así como la perspectiva personal sobre el proyecto.

Capítulo 2

Planificación

En este apartado se describe la organización del proyecto: la metodología utilizada para guiar el desarrollo del sistema *software*, las estimaciones temporales y monetarias, con la posterior comparación con los costes reales, y el análisis de los posibles riesgos que puedan ocurrir durante el proyecto.

2.1. Metodología de trabajo

La metodología utilizada para la organización y planificación de este proyecto es ASAP (Agile Student Academic Projects) [25], basada en los marcos de trabajo ágiles (especialmente Scrum). Esta metodología busca abordar los objetivos de aprendizaje del TFG: planificación del proyecto, consolidación de sus antecedentes, desarrollo y aceptación del producto, y comunicación (tanto oral como escrita) del trabajo realizado.

ASAP redefine los roles, eventos y artefactos de Scrum. Por un lado, los roles definen a los participantes en el proyecto: el estudiante que desarrolla el TFG, los tutores que guían al alumno, la comunidad formada por otros estudiantes preparando su TFG, y el tribunal que evalúa el TFG.

Los eventos que define ASAP permiten llevar un seguiemiento del TFG y asegurar la interacción continua entre el estudiante y los tutores, estableciendo un ritmo de trabajo sostenido durante todo el TFG. ASAP estructura los TFG en *sprints*, períodos de tiempo con una duración máxima de un mes en los que se planifican y abordan un conjunto de objetivos concreto. En cada *sprint* se llevan a cabo varias ceremonias: la reunión de inicio, en la que se establecen los objetivos del *sprint* y las tareas necesarias para alcanzarlo; las reuniones de sincronización, realizadas semanalmente para llevar un seguimiento del TFG y comunicar posibles bloqueos; la comunicación de progresos al final del *sprint*, donde el estudiante presenta su TFG frente a los profesores y la comunidad como si fuese la presentación frente al tribunal; y la retrospectiva, en la que el alumno, los profesores y la comunidad reflexionan sobre la calidad del proceso seguido, indicando de forma anónima los aspectos positivos y negativos, así como formas de mejorar.

Finalmente, los artefactos que propone ASAP son el incremento, que reúne los resultados consolidados hasta el final del *sprint*; y la retroalimentación o *feedback* que generan los tutores sobre el incremento.

ASAP requiere un entorno tecnológico compuesto por tres elementos: un espacio de trabajo compartido entre el estudiante y el tutor, para facilitar la comunicación y la transmisión de recursos; un tablero de proyecto KanBan, para organizar las tareas; y un cuaderno de trabajo,

donde el estudiante lleva un registro de las tareas realizadas y el tiempo invertido en las mismas, con el objetivo de mejorar la gestión del tiempo.

2.2. Planificación temporal

Como se especifica en el apartado 2.1, la metodología utilizada en este proyecto es ASAP, que organiza al mismo en *sprints*. En cada *sprint* se abordan un conjunto de objetivos que se deciden en la reunión de inicio (al comienzo del *sprint*). Para realizar la planificación temporal del proyecto, se establecen objetivos a mayores para cada *sprint* y se concretan al inicio de cada uno de ellos.

Los *sprints* tienen una duración de un mes excepto el último, que dura dos semanas (tres si se presenta en la extraordinaria). Desde marzo hasta mayo cada semana dedican 30 horas a otras asignaturas, y tomando como referencia el horario de jornada completa (40 horas a la semana), se dispone de 10 horas a la semana, con un total de 10 horas $\cdot$ 4 semanas $\cdot$ 3 meses = 120 horas. En junio y julio se estima tener más tiempo para dedicar al proyecto, en torno a unas 25 horas por semana, por lo que el total de horas será de 120 horas + (25 horas $\cdot$ 6 semanas) = 270 horas, 295 horas si se presenta en la extraordinaria, que es próximo al límite de 300 horas que marca la restricción RE-01 1.2.1. Entonces, el alcance del proyecto se limita para ajustarse a estas 270 horas, tomando las 25 horas de la semana extra como sobrecarga, obteniendo como duración total del proyecto 295 horas.

Cabe destacar que en todos los *sprints* tras el primero se utilizará la retroalimentación generada por los tutores para mejorar el producto, por lo que se añadirá a los objetivos de ese *sprint* en la reunión de inicio.

La estimación de la duración del proyecto se llevará a cabo mediante el método de puntos de Caso de Uso.

2.2.1. Sprint 1

Este *sprint* comienza el 27 de febrero de 2023 y termina el 24 de marzo de 2023, pudiendo destinar 40 horas de esfuerzo. En este *sprint* se definen el alcance y objetivos del proyecto, así como las restricciones que lo limitan. Tras esto, se lleva a cabo la planificación tanto temporal como presupuestaria, definiendo así qué objetivos se abordarán en cada *sprint*. Se comenzará a estudiar las tecnologías que se pueden utilizar para desarrollar al agente inteligente.

Al inicio del *sprint* se ha decidido el horario en el que se llevarán a cabo las reuniones que propone ASAP, detalladas en el apartado 2.1:

- La reunión de inicio se realizará el primer día (lunes) de cada *sprint* a las 18:30.
- Las reuniones de sincronización se llevarán a cabo el lunes a las 18:30 de cada semana.
- La comunicación de progresos se realizará el jueves a las 16:30 de la última semana del *sprint*.
- La restrospectiva se llevará a cabo al terminar la comunicación de progresos.

2.2.2. Sprint 2

Este *sprint* comienza el 27 de marzo de 2023 y termina el 28 de abril de 2023 (no se tiene en cuenta el descanso por Semana Santa). En este *sprint* se investigará cómo definir e implementar los actuadores y los sensores. Por otro lado, se explicará la metodología de trabajo y los antecedentes del proyecto, y se definirán los casos de uso, requisitos funcionales y no funcionales.

2.2.3. Sprint 3

Este *sprint* comienza el 1 de mayo de 2023 y termina el 26 de mayo de 2023. Los objetivos de este *sprint* se centrarán en diseñar y desarrollar el modelo del agente, con la finalidad de obtener un agente completo básico capaz de aprender. Al final del *sprint* anterior se encontró una solución más eficiente a la hora de diseñar los sensores, por lo que en este *sprint* se implementará esta solución. Otro cambio a tener en cuenta es una reducción del número de horas disponibles para este *sprint*, al tener que dedicar más horas a las prácticas hasta finalizar el mes, teniendo en cuenta que estas horas estarán disponibles en junio. Se estima que se podrán dedicar unas cinco horas por semana al proyecto, con un total de veinte horas. Por ello, la implementación del modelo del agente se aplazará al *sprint* siguiente.

2.2.4. Sprint 4

Este *sprint* comienza el 29 de mayo de 2023 y termina el 23 de junio de 2023. En este *sprint* se llevará a cabo el entrenamiento del agente con los ajustes que sean necesarios, documentando todo el proceso. Se definirán los requisitos de información, se realizará el diseño del sistema *software* y se definirán las métricas que se utilizarán para evaluar al agente.

Debido a los cambios del *sprint* anterior, en este se implementará el modelo del agente para obtener una versión completa del mismo. Para evitar la necesidad de horas extra, la entrega del TFG se realizará el día 14 de julio de 2023, para la convocatoria extraordinaria, obteniendo veinticinco horas de esfuerzo frente a las veinte perdidas el *sprint* anterior.

2.2.5. Sprint 5

Este *sprint* comienza el 26 de junio de 2023 y termina el 14 de julio de 2023. En este *sprint* se llevarán a cabo las pruebas de caja negra, y se redactarán la conclusiones junto con los manuales de instalación y usuario. Debido a los cambios de *sprints* anteriores, en este se terminará el aprendizaje del agente inteligente, cerrando la experimentación y la discusión de resultados.

2.3. Presupuestos

El presupuesto es una estimación de los gastos que supondrá el proyecto. Para calcular estos gastos se tendrán en cuenta los recursos utilizados y la cantidad de tiempo o unidades que se estiman utilizar, así como el esfuerzo humano que requiere el proyecto.

Personal Se tienen en cuenta los roles de la tabla 2.1. Para definir el sueldo bruto anual se ha tenido en cuenta el nivel de experiencia (junior) y la media en España [9]. El sueldo por horas se obtiene al dividir el sueldo anual entre 2080 horas por año (52 semanas por año y 40 horas por semana).

Rol	Tareas	Sueldo anual	Sueldo por hora
Gestor	Planificación, coordinación y supervisión	35000€	16.83€
Científico de datos	Investigación, modelado y ajuste del agente inteligente	29335€	14.32€
Analista	Análisis de requisitos y diseño	25313€	12.17€
Programador	Implementación del agente inteligente y la extensión Puente, y pruebas	20650€	9.93€

Tabla 2.1: Roles en el proyecto.

Hardware Se tiene en cuenta el uso del hardware en el proyecto respecto al tiempo de vida útil del dispositivo. Es necesario un equipo de alto rendimiento para poder entrenar al agente (1250€). Se estiman unas 50 horas de uso semanales durante 4 años de vida útil, obteniendo un total de 10400 horas de vida útil y 12.02 céntimos por hora de uso.

Software Todas las herramientas y librerías son gratuitas, pero el videojuego es de pago.

Conectividad Se tiene en cuenta el uso de la red destinado al proyecto.

Otros gastos Por ejemplo, el consumo de gasolina de un viaje ida y vuelta para acudir a algunas reuniones.

Descripción	Coste / Unidad	Unidades	Coste total
Gestor	16.83€/h	59h (20%)	992.97€
Científico de datos	14.32€/h	118h (40%)	1689.76€
Analista	12.17€/h	59h (20%)	718.03€
Programador	9.93€/h	59h (20%)	585.87€
Ordenador (sin aprendizaje)	12.02cént/h	295h	35.46€
Ordenador (aprendizaje)	12.02cént/h	50h	6.01€
Copia de BloonsTD6	10.99€/u	1u	10.99€
Conexión a Internet (ADSL)	28.95€/mes (4.02cént/h)	295h	11.86€
Viajes por reuniones	1.639€/L · 6.5L/100Km=10.65cént/Km	318Km	33.87€
Total			**4084.82€**

Tabla 2.2: Tabla de presupuestos

2.4. Gestión de riesgos

Los riesgos son problemas potenciales que afectan a futuros acontecimientos, y que suponen cambios en el proyecto, al tener que decidir cómo solventarlos. Los riesgos tienen unas causas

que los provocan y un impacto sobre el proyecto. Para hacerlos frente, se de analizar los posibles riesgos que puedan afectar al proyecto y desarrollar planes de prevención y contingencia.

Para clasificar y evaluar los riesgos se estudia la probabilidad o frecuencia de que ocurran y el impacto sobre el proyecto que suponen. La probabilidad se define en seis tramos: mínimo (1 %), muy baja (10 %), baja (30 %), media (50 %), alta (70 %) y muy alta (90 %). Para el impacto se tendrá en cuenta el número de horas que supone su solución. El producto de estos dos valores, denominado severidad, permitirá evaluar cómo afectarán los riesgos a las estimaciones del proyecto.

R-01	**Actualización del videojuego**
Descripción	La actualización de BloonsTD6 provoca que las direcciones de memoria a los parámetros del juego cambien, por lo que hay que obtenerlas de nuevo manualmente.
Probabilidad	El juego se suele actualizar cada tres o cuatro meses, por lo que la probabildad es muy alta
Impacto	Tres horas por mapa. El agente interactúa con dos mapas, por lo que seis horas.
Severidad	Tiempo: $0.9 \cdot 6\text{h} = \mathbf{5.4h}$. Coste de ordenador: $12.02\text{cént/h} \cdot 5.4\text{h} = 0.65€$. Coste de personal (programador): $9.93€/\text{h} \cdot 5.4\text{h} = 53.62€$. Coste final: $0.65€ + 53.62€ = \mathbf{54.27€}$

Tabla 2.3: Riesgo R-01: Actualización del videojuego.

R-02	**Incapacidad de uso del ordenador**
Descripción	No poder utilizar el ordenador donde se desarrolla el proyecto porque no encienda, se estropee la batería, etc
Probabilidad	Muy baja
Impacto	El trabajo está respaldado en la nube, por lo que no se perdería nada. La implementación se retrasaría hasta reparar el ordenador (una semana en el peor de los casos).
Severidad	Antes de mayo: $0.1 \cdot 10\text{h} = \mathbf{1h}$. Después de mayo: $0.1 \cdot 25\text{h} = \mathbf{2.5h}$.

Tabla 2.4: Riesgo R-02: Incapacidad de uso del ordenador.

R-03	**Reducción de velocidad de entrenamiento**
Descripción	El ordenador puede que no aguante la velocidad del juego 25 veces acelerada durante largos periodos de entrenamiento, debido a un uso excesivo de los componentes, que provocase que se congelase y se perdiese el entrenamiento.
Probabilidad	Media
Impacto	Reducir la velocidad de entrenamiento a 10 veces más rápida
Severidad	Tiempo: $\frac{25}{10} \cdot 50\text{h} = \mathbf{125h}$. Coste: $125\text{h} \cdot 12.02\text{cént/h}$ (coste ordenador) = $\mathbf{15.03€}$.

Tabla 2.5: Riesgo R-03: Reducción de velocidad de entrenamiento.

R-04	**Bloqueo de las extensiones**
Descripción	Ninja Kiwi puede decidir bloquear las extensiones en BloonsTD6, imposibilitando el uso de la extensión Puente y FasterForward (que permite acelerar el entrenamiento 25 veces más rápido, apartado 3.1.2).
Probabilidad	Las extensiones han sido válidas desde poco después del inicio del juego, por lo que la probabilidad es mínima
Impacto	Se utilizaría el método de las direcciones de memoria, y el uso del ordenador para el aprendizaje sería 25 veces mayor
Severidad	Tiempo: $0.01 \cdot 25 \cdot 50\text{h} = \mathbf{12.5h}$. Coste: $12.5\text{h} \cdot 12.02\text{cént/h}$ (coste ordenador) $= \mathbf{1.50\,€}$.

Tabla 2.6: Riesgo R-04: Bloqueo de las extensiones.

Así, el coste total por prevención de riesgos es de 70.80 €, y el tiempo de prevención será de 7.9 horas, que se pueden considerar como parte de la sobrecarga.

2.5. Balance temporal y económico

2.5.1. Sprint 1

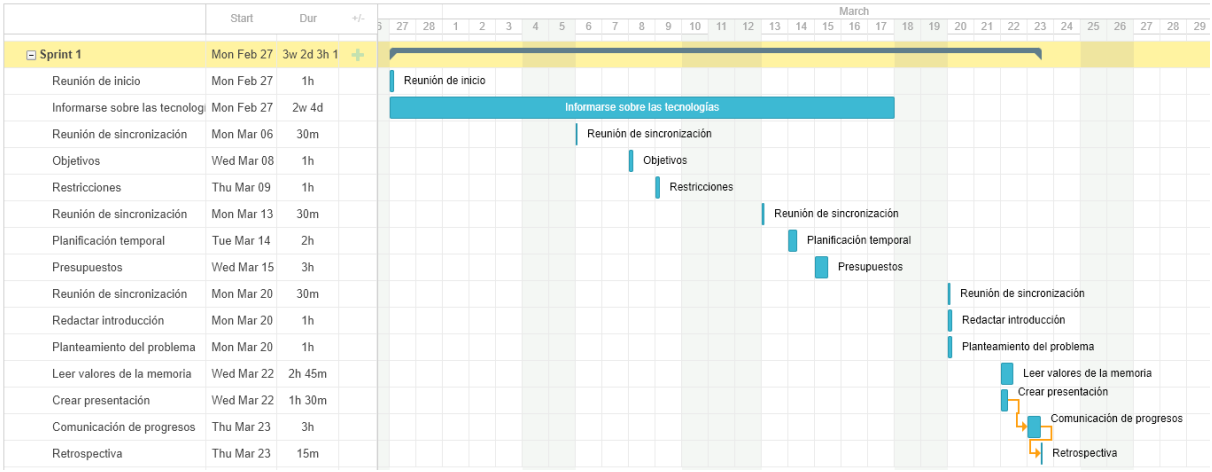


Figura 2.1: Diagrama de Gantt del *sprint* 1

Este *sprint* se han llevado a cabo 44 horas y 20 minutos de trabajo, un poco más de las 40 horas de esfuerzo que se preveían disponibles. Se han completado todos los objetivos planificados, y los plazos definidos para las tareas se han cumplido satisfactoriamente, en parte gracias a la flexibilidad que ha dado la tarea "Informarse sobre las tecnologías". Entonces, el balance de gastos es el siguiente:

Descripción	Coste / Unidad	Unidades	Coste total
Personal: Gestor	16.83 €/h	15.75h	265.07 €
Personal: Científico de datos	14.32 €/h	25.83h	369.89 €
Personal: Programador	9.93 €/h	2.75h	27.31 €
Ordenador	12.02cént/h	44.33h	5.33 €
Copia de BloonsTD6	10.99 €/u	1u	10.99 €
Conexión a Internet (ADSL)	4.02cént/h	44.33h	1.78 €
Viajes por reuniones	10.65cént/Km	106Km	11.29 €
Total			**691.66 €**

Tabla 2.7: Balance económico del *sprint* 1

2.5.2. Sprint 2

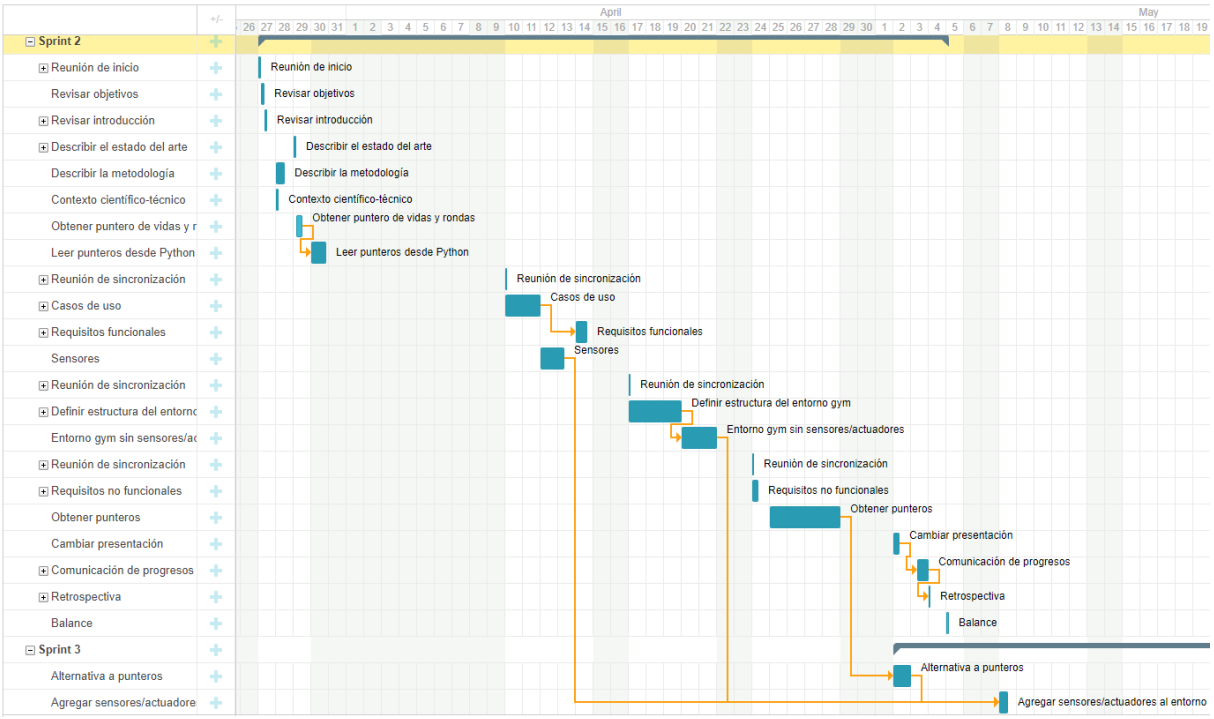


Figura 2.2: Diagrama de Gantt del *sprint* 2

El desarrollo de este *sprint* ha sido el esperado excepto al final, ya que el juego se actualizó. Esto se había contemplado en el riesgo R-01 (apartado 2.4), pero mientras se obtenían los nuevos punteros, se descubrió otra solución más óptima: el uso de extensiones del videojuego. Entonces, se ha decidido añadir como objetivo del *sprint* siguiente la creación de esta extensión. La duración del *sprint* ha sido de 49 horas, mayor de la estimada porque se ha aprovechado algunos días de la Semana Santa para adelantar trabajo. La duración hasta el momento es de 93 horas y 20 minutos.

Descripción	**Coste / Unidad**	**Unidades**	**Coste total**
Personal: Gestor	16.83 €/h	14h	235.62 €
Personal: Científico de datos	14.32 €/h	11h	157.52 €
Personal: Analista	12.17 €/h	10h	121.7 €
Personal: Programador	9.93 €/h	14h	139.02 €
Ordenador	12.02cént/h	49h	5.89 €
Conexión a Internet (ADSL)	4.02cént/h	49h	1.97 €
Viajes por reuniones	10.65cént/Km	106Km	11.29 €
Subtotal *sprint 2*			**673.01 €**
Coste de anteriores *sprints*			691.66 €
Total			**1364.67 €**

Tabla 2.8: Balance económico del *sprint* 2

2.5.3. Sprint 3

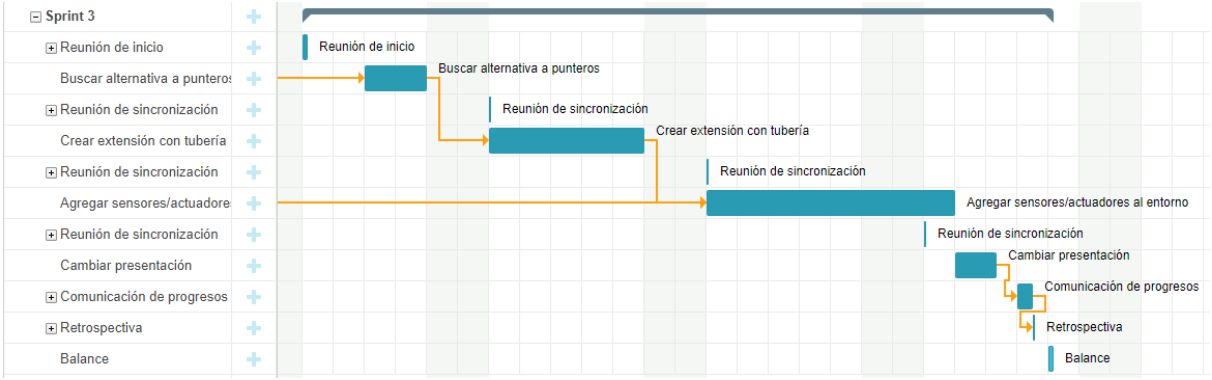


Figura 2.3: Diagrama de Gantt del *sprint* 3

Este *sprint* ha sufrido cambios considerables debido a varios factores. Por un lado, he tenido que dedicar más tiempo a las prácticas y otras asignaturas, dejando más tiempo disponible en junio. Por otro lado, se ha aplazado a este *sprint* el desarrollo de la extensión Puente. Como consecuencia, en este *sprint* se ha centrado en implementar los sensores e implementarlos al entorno junto con los actuadores. El esfuerzo dedicado ha sido de 23 horas, un poco más de lo que se estimaba disponible, haciendo un total de 116 horas.

Descripción	**Coste / Unidad**	**Unidades**	**Coste total**
Personal: Gestor	16.83 €/h	8h	134.64 €
Personal: Científico de datos	14.32 €/h	4h	57.28 €
Personal: Programador	9.93 €/h	11h	109.23 €
Ordenador	12.02cént/h	23h	2.76 €
Conexión a Internet (ADSL)	4.02cént/h	23h	0.92 €
Viajes por reuniones	10.65cént/Km	106Km	11.29 €
Subtotal *sprint 3*			**316.12 €**
Coste de anteriores *sprints*			1364.67 €
Total			**1680.79 €**

Tabla 2.9: Balance económico del *sprint* 3

2.5.4. Sprint 4

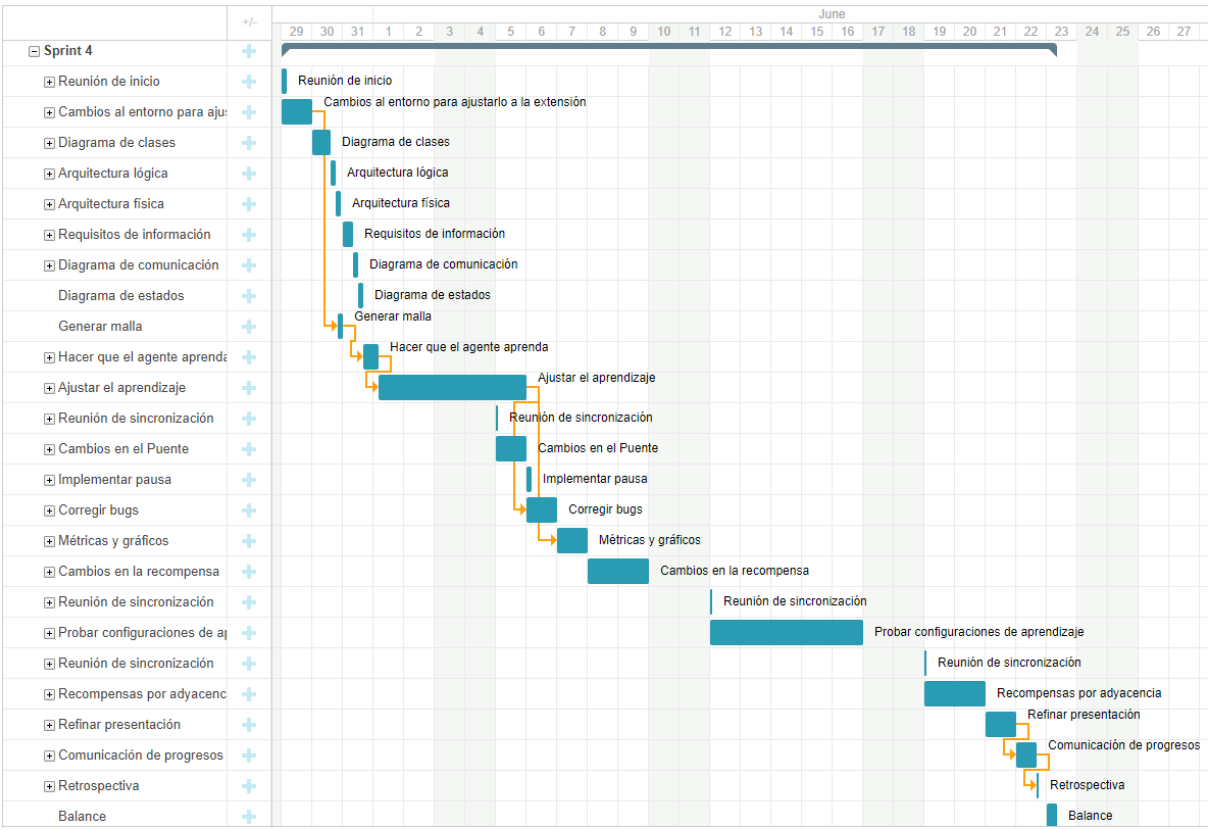


Figura 2.4: Diagrama de Gantt del *sprint* 4

Como consecuencia del *sprint* anterior, los objetivos de este *sprint* han incluído el desarollo del modelo del agente. El resto de objetivos se han podido cumplir, exceptuando el aprendizaje del agente que, aunque ha comenzado, se debe continuar en el *sprint* siguiente. El esfuerzo

dedicado a este *sprint* es de 99 horas, una menos de la esperada, haciendo un total de 215 horas. Sin embargo, esto no significa que el proyecto esté adelantado a la planificación, ya que el aprendizaje del agente se realiza en los períodos en los que no se utiliza el ordenador (porque como simula los periféricos no se pueden realizar otras tareas), marcando los tiempos en varias ocasiones.

Tras la comunicación de progresos se estropeó el cargador del portátil, por lo que no puedo avanzar en el entrenamiento hasta que reciba uno nuevo (unos dos días laborales). Esto se ha previsto en el riesgo R-02 (apartado 2.4), pero en principio sólo se debería de perder la mañana del lunes.

Descripción	**Coste / Unidad**	**Unidades**	**Coste total**
Personal: Gestor	16.83€/h	14h	235.62€
Personal: Científico de datos	14.32€/h	62.5h	895€
Personal: Analista	12.17€/h	9.5h	115.62€
Personal: Programador	9.93€/h	13h	129.09€
Ordenador (sin entrenamiento)	12.02cént/h	99h	11.90€
Ordenador (entrenamiento)	12.02cént/h	20h	2.40€
Conexión a Internet (ADSL)	4.02cént/h	99h	3.98€
Viajes por reuniones	10.65cént/Km	106Km	11.29€
Subtotal *sprint 4*			**1404.90€**
Coste de anteriores *sprints*			1680.79€
Total			**3085.69€**

Tabla 2.10: Balance económico del *sprint* 4

2.5.5. Sprint 5

Este último *sprint* ha consistido en terminar el aprendizaje del agente inteligente y cerrar el pruyecto. Gracias a la semana extra, se ha podido mantener un ritmo estable de trabajo, resultando en 62 horas de esfuerzo y cerrando el proyecto con 277 horas, 18 horas menos que las 295 propuestas.

El presupuesto del proyecto era de 4084.82€, por lo que el coste real es de 127.48€ menos. Esto se debe principalmente a las 18 horas menos de esfuerzo necesario que las estimadas al inicio.

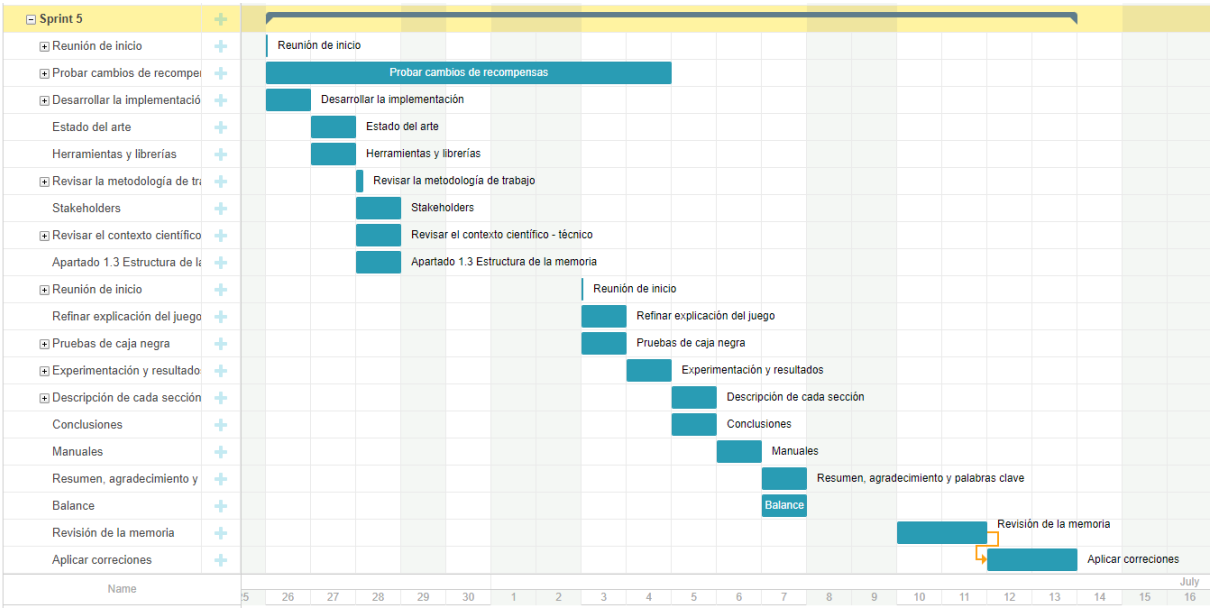


Figura 2.5: Diagrama de Gantt del *sprint* 5

Descripción	**Coste / Unidad**	**Unidades**	**Coste total**
Personal: Gestor	16.83 €/h	11h	185.13 €
Personal: Científico de datos	14.32 €/h	36h	515.52 €
Personal: Analista	12.17 €/h	4h	48.68 €
Personal: Programador	9.93 €/h	11h	109.23 €
Ordenador (sin entrenamiento)	12.02cént/h	62h	7.45 €
Ordenador (entrenamiento)	12.02cént/h	26h	3.13 €
Conexión a Internet (ADSL)	4.02cént/h	62h	2.49 €
Subtotal *sprint 5*			**871.63 €**
Coste de anteriores *sprints*			3085.69 €
Total			**3957.32 €**

Tabla 2.11: Balance económico del *sprint* 5

Capítulo 3

Antecedentes

En esta sección se analiza el entorno en el que se desarrolla el proyecto, se indican las herramientas y librerías que se utilizan, junto con una explicación del videojuego BloonsTD6. Tras esto, se profundiza en las tecnologías utilizadas, y se estudian otros proyectos relacionados para adquirir otras perspectivas y compararlas con este proyecto.

3.1. Entorno de negocio

Desde el inicio de los videojuegos hasta la actualidad, la inteligencia artifical en los videojuegos se ha centrado en dos áreas: *pathfinding* y máquinas de estados finitos. El *pathfinding*, o búsqueda de caminos, se encarga de definir a qué zonas puede acceder o no un personaje o un objeto móvil en un videojuego, así como de encontrar qué ruta es la más óptima desde su posición hasta su destino (fig. 3.1). Por otro lado, las máquinas de estados finitos (fig. 3.2) se encargan de cambiar de comportamiento según el estado del entorno, simulando el proceso de toma de decisiones de los humanos. El agente de este proyecto se centra en este área.

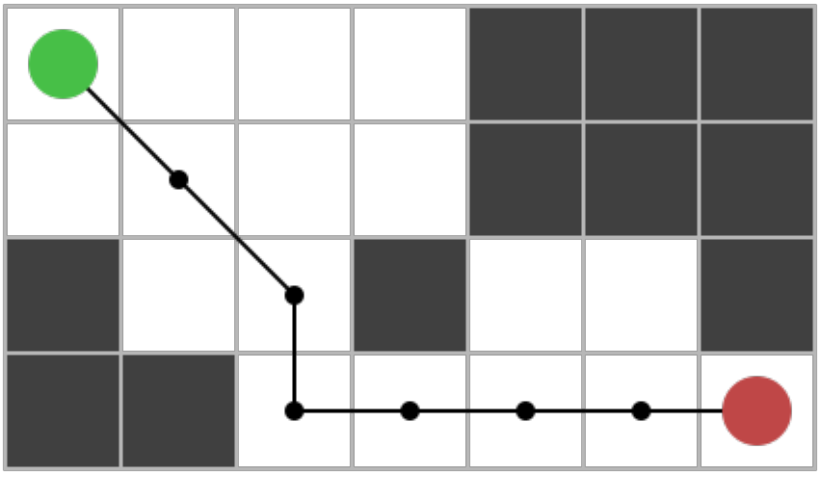


Figura 3.1: Ejemplo de búsqueda del camino desde el punto verde hasta el rojo.

Mientras que estas dos áreas pertenecen al videojuego en sí, el desarrollo de videojuegos

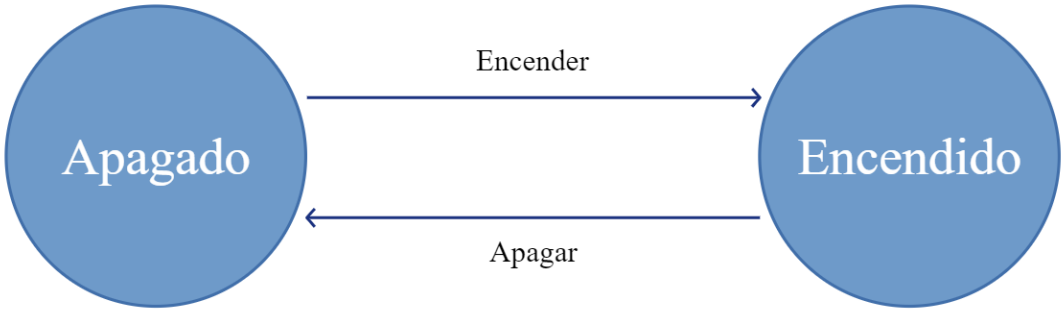


Figura 3.2: Ejemplo de máquina de estados finitos de una bombilla.

también está evolucionando tras la aparición de las inteligencias generativas estos dos últimos años, como "Stable Diffusion" [36], que permite generar imágenes a partir de texto, o "ChatGPT" [10], que se puede utilizar como asistente en la programación.

3.1.1. Interesados (Stakeholders)

Un interesado o *stakeholder* es un individio, grupo u organización que participa activamente en el proyecto, está afectada por el proceso o sus resultados, o tiene la capacidad de influir en el proceso o sus resultados. Los interesados pueden ser miembros del equipo de proyecto, de la organización que desarrolla el proyecto, o ser ajenos tanto del equipo como de la organización.

Cada interesado está caracterizado por un nombre, el rol que ejerce en el proyecto, los requisitos que exige al proyecto, el grado de influencia que ejerce, los intereses o espectativas que tiene en el proyecto, la pertenencia a la organización donde se desarrolla el proyecto (si pertenece, es interno, si no, externo) y la posición frente al proyecto (a favor, neutral o en contra).

Nombre	**Gustavo Cortés Jiménez**
Rol	Estudiante
Requisitos	Diseñar un plan de proyecto. Analizar, diseñar y desarrollar el agente inteligente.
Influencia	Alta
Interés	Éxito en el TFG. Adquirir conocimiento sobre los agentes inteligentes y el aprendizaje por refuerzo.
Interno/ externo	Interno
Posición	A favor

Tabla 3.1: Interesado Gustavo Cortés Jiménez.

Nombre	**Aníbal Bregón Bregón, Jorge Silvestre Vilches**
Rol	Tutores
Requisitos	El alumno es capaz de gestionar el proyecto y desarrollar un producto de calidad.
Influencia	Alta
Interés	Éxito en el TFG. El alumno cumple los plazos y objetivos del proyecto definidos según la metodología ASAP.
Interno/ externo	Interno
Posición	A favor

Tabla 3.2: Interesado Aníbal Bregón Bregón, Jorge Silvestre Vilches.

Nombre	**UVA**
Rol	Universidad
Requisitos	Cumplimiento de los requisitos y normativas para el TFG.
Influencia	Media
Interés	Promover la investigación en el campo de la Inteligencia Artificial. Contribuir a la reputación y prestigio de la universidad.
Interno/ externo	Interno
Posición	A favor

Tabla 3.3: Interesado UVA.

Nombre	**Ninja Kiwi**
Rol	Empresa propietaria de BloonsTD6.
Requisitos	Cumplir los Términos y Condiciones [39].
Influencia	Alta
Interés	Un agente que ayude a evaluar nuevas características de BloonsTD6. No utilizar la herramienta para obtener una ventaja sobre otros jugadores.
Interno/ externo	Externo
Posición	Neutral

Tabla 3.4: Interesado Ninja Kiwi.

Nombre	**Jugadores**
Rol	Usuarios de BloonsTD6.
Requisitos	Facilidad de uso.
Influencia	Baja
Interés	Encontrar nuevas estrategias. Tener un compañero en el modo cooperativo sin necesidad de una conexión a Internet.
Interno/ externo	Externo
Posición	A favor

Tabla 3.5: Interesado Jugadores.

3.1.2. Herramientas y librerías

Las herramientas y librerías utilizadas en este proyecto se dividen en dos grupos: aquellas para facilitar la gestión del proyecto y la comunicación entre los tutores, alumno y comunidad; y las destinadas a la implementación del agente, principalmente librerías de Python. La redacción de la memoria se ha realizado a través de Overleaf [29], un editor de LaTeX en línea.

Herramientas y librerías para la implementación

El lenguaje de programación escogido para desarrollar al agente es Python, debido a la gran cantidad de librerías disponibles que han permitido agilizar la implementación de los mecanismos de interacción del agente, además de poder utilizar la librería anteriormente nombrada *stable-baselines3*. Concretamente, se ha utilizado Anaconda [4], una distribución de Python orientada a la computación científica, que facilita la gestión e implementación de librerías. Anaconda incluye Visual Studio Code [41], el editor de código utilizado para desarrollar los *scripts*. Para crear la extensión Puente con la que se comunica el agente, se ha utilizado el lenguaje de programación C#, que es el mismo con el que está escrito BloonsTD6, a través del editor de código Visual Studio [40] que incluye el compilador de este lenguaje.

La librería *pydirectinput* [21] facilita la interación del agente con el videojuego, ya que ofrece utilidades que permiten simular el teclado y el ratón. De esta forma, el agente es capaz de seleccionar torres con atajos del teclado, y clicar en la posición donde quiera colocarla. Estas torres se almacenan en matrices, que se definen y gestionan a través de la librería *numpy* [27].

El modelo del agente, que es el componente que toma las decisiones, sigue la estructura que define la librería *gymnasium* [18], que se utiliza como base de los algoritmos de aprendizaje que facilita *stable-baseline3*, librería que contiene un conjunto de implementaciones de lenguajes de algoritmos de aprendizaje por refuerzo, facilitando considerablemente la implementación de los mismos.

La herramienta MelonLoader [14] permite la ejecución de extensiones en BloonsTD6. Junto con esta herramienta, se utilizan otras dos extensiones: *BTD-Mod-Helper* [37], que se encarga de gestionar las extensiones y ofrece una API para la creación de las mismas; y *faster-forward* [16], que permite acelerar la ejecución del videojuego.

Herramientas para la gestíón

La plataforma Teams [26] ha facilitado un espacio de trabajo virtual a través de la cual se ha llevado la comunicación oral y escrita entre el estudiante y los tutores, así como con los miembros de la comunidad. También se ha utilizado esta plataforma para almacenar y organizar los artefactos generados en el proyecto.

Se ha utilizado Trello [17] para administrar las tareas del proyecto. Esta herramienta ofrece un tablero Kanban donde se organizan tarjetas que respresentan a las tareas, asignándolas un nombre, una descripción, listas de subtareas, etc.

3.1.3. Bloons Tower Defense 6

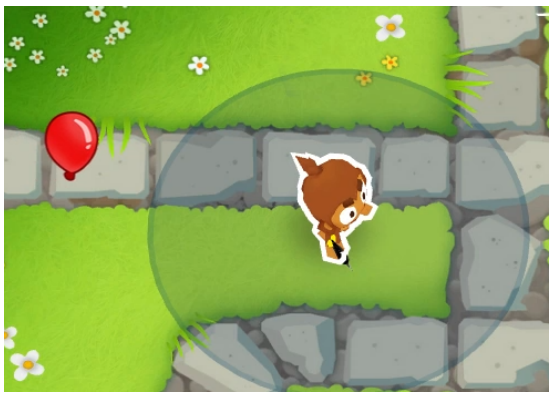
En este proyecto se estudia y desarrolla un agente capaz de jugar al videojuego Bloons Tower Defense 6 (BloonsTD6), de Ninja Kiwi. Este juego pertenece al género de "defensa de la torre", el cual consiste en defenderse de oleadas de enemigos colocando "torres" en un mapa, que atacan automáticamente a estos enemigos. Si estos enemigos llegan al final del recorrido, provocarán que el jugador pierda vidas. Si se pierden todas las vidas, se acaba la partida. Para ganar, el jugador deberá superar todas las rondas.

Figura 3.3: Partida de BloonsTD6

Torres

En BloonsTD6, las torres son principalmente monos y los enemigos, globos llamados "Bloons". Las torres se colocan en el mapa por un precio y disparan automáticamente a los globos que estén dentro de su alcance, que se indica en el juego mediante un círculo oscuro alrededor de la torre (fig. 3.4). Otro tipo de torres son las de apoyo, cuyo objetivo es aportar utilidad, ya sea

mejorando otras torres, ralentizando a los Bloons, etc. Un caso especial son los héroes, un tipo de torre que se mejora a sí misma según avanzan las rondas; son muy poderosas, pero sólo se puede colocar una por partida.



(a) El globo no está en el alcance del mono, por lo que no lo ataca.

(b) El mono revienta el globo porque está en su alcance.

Figura 3.4: Alcance de las torres

Al clicar sobre una torre, se abre un menú (fig. 3.5) para gestionarla. En la parte superior se muestra un contador con el número de capas que ha reventado la torre en total. Debajo de la imagen de la torre, se puede escoger cómo va a elegir su objetivo: apuntar al primero, es decir, el globo que esté más cerca de la salida; al último, que será el que esté más cerca del inicio; al enemigo más cercano a la torre, o al más fuerte, es decir, el globo que tenga más capas. En la parte inferior se muestra un botón para vender la torre por el 70 % de su precio original.

Figura 3.5: Menú de un mono lanzadardos.

En la parte central del menú se muestran las mejoras de la torre. Cada torre tiene tres ramas de mejoras, que la atribuyen de nuevas características o refuerzan las que ya tiene, como se ve en la figura 3.6. Cada mejora sólo se puede obtener si se ha comprado la mejora anterior de su rama, y su precio suele ser mayor cuanto más profunda se encuentre en la rama. Las mejoras se pueden combinar siguiendo dos reglas: sólo se pueden tener mejoras de dos ramas, y de una de las dos únicamente se pueden tener las dos primeras mejoras; es decir, que lo máximo que puede estar una torre mejorada es una rama con las cinco mejoras y otra con dos: por ejemplo, en la figura 3.7 se muestra un mono lanzadardos 520, es decir, 5 mejoras de la primera rama, 2 de la segunda y 0 de la tercera.

Figura 3.6: Árbol de mejoras de un mono lanzadardos.

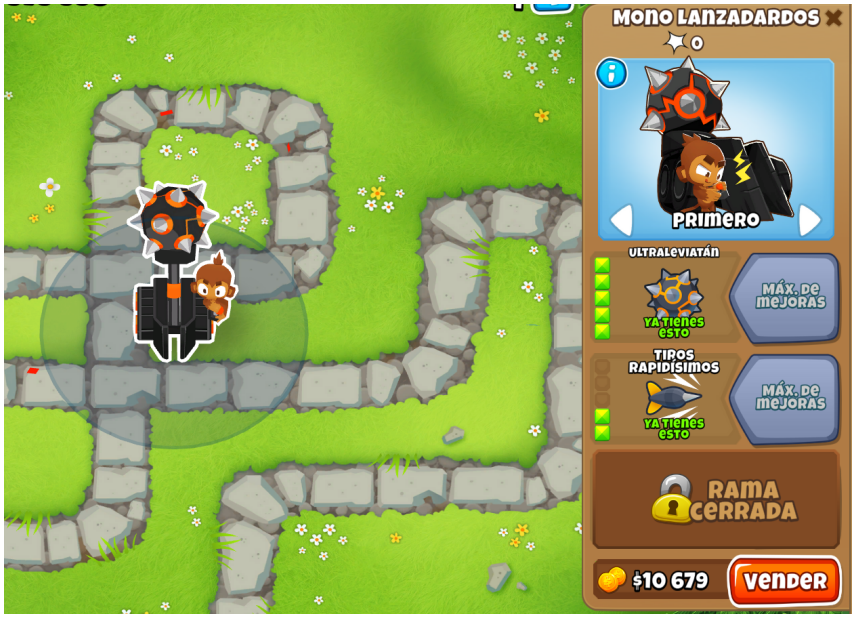


Figura 3.7: Mono lanzadardos con las mejoras 520.

Bloons

Los Bloons son los enemigos en BloonsTD6. Están compuestos por capas de forma recursiva, comenzando por el globo rojo, es decir, una sóla capa que si se revienta, desaparece el globo. El siguiente es el globo azul, que al reventarlo libera un globo rojo; después el globo verde, que contiene uno azul, y así sucesivamente (fig. 3.8). Hay Bloons que contienen varios globos dentro de ellos, liberándolos juntos al reventar.

Figura 3.8: Capas de un globo verde

Las capas tienen características propias: de base, cada capa tiene una velocidad asignada, siendo en general más rápidos los Bloons con más capas. Algunas capas otorgan inmunidades, por ejemplo, los globos negros son inmunes a las explosiones y los emplomados son resistentes a la mayoría de ataques, menos las explosiones. Por otro lado, existen dos tipos de ventajas (fig. 3.9) que pueden tener todos los globos (incluso simultáneamente): el camuflaje, que sólo podrán ver algunas torres con mejoras concretas; y la regeneración, que permite al Bloon recuperar capas que ha perdido, hasta su capa original.

(a) Bloon con camuflaje

(b) Bloon con regeneración

Figura 3.9: Ventajas generales de los globos

Existe un tipo especial de Bloons, los dirigibles, que a efectos prácticos actúan como una capa más: la "capa MOAB" (obtiene el nombre del dirigible más sencillo, el MOAB, fig. 3.10). Esta capa se caracteriza por aguantar daño sin reventar (por ejemplo, un MOAB resiste 200 de daño

antes de reventar). Además, son más resistentes a las desventajas que causan algunos monos, como las ralentizaciones (se necesitan mejoras concretas para ralentizar dirigibles).

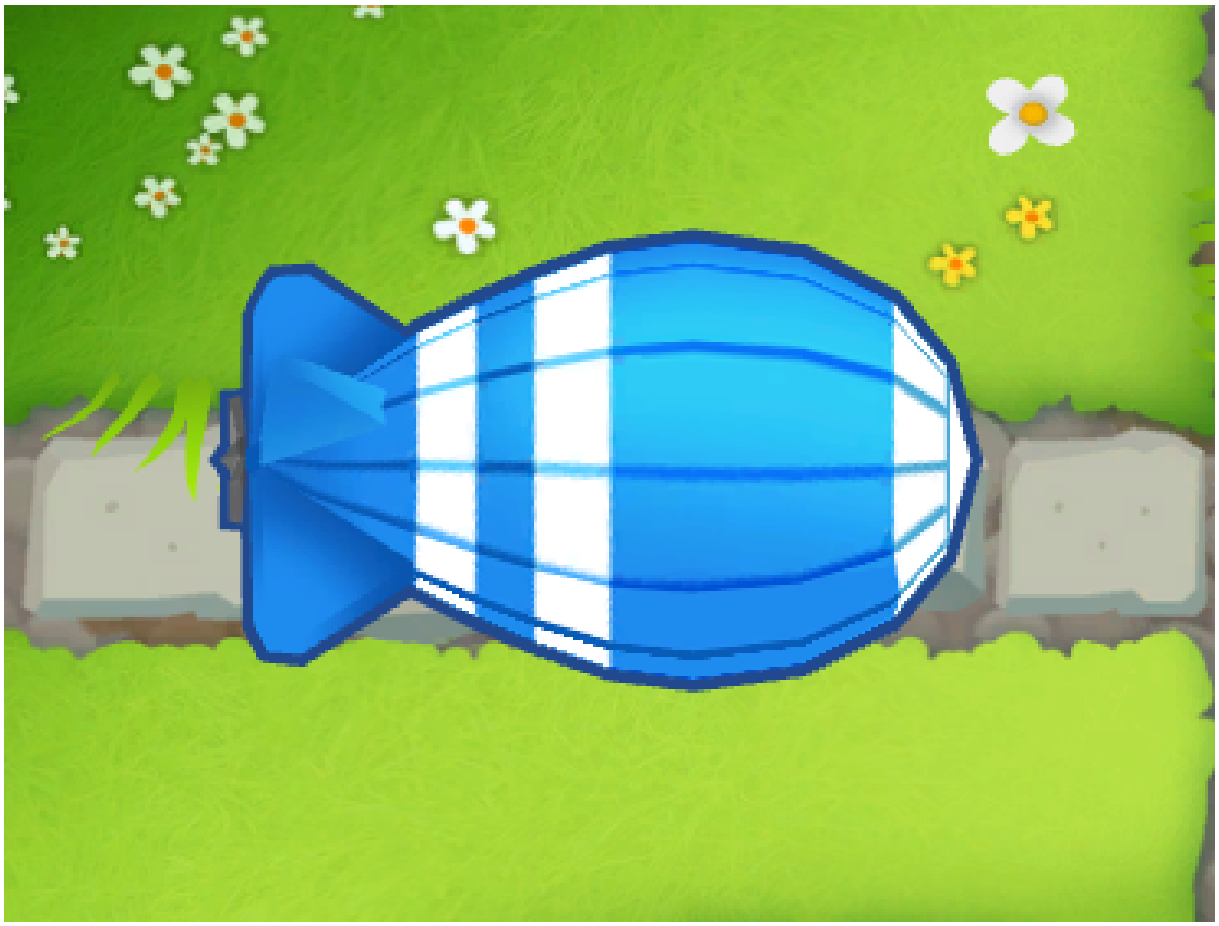


Figura 3.10: MOAB, el dirigible más básico

Mapas y modos de juego

BloonsTD6 presenta 69 mapas divididos en cuatro categorías según su dificultad (principante, intermedio, avanzado y experto). Cada mapa contiene uno o varios caminos por donde avanzan los Bloons, y dispone de terreno y agua donde se pueden colocar las torres, así como obstáculos para entorpecer su desempeño.

Una vez escogido el mapa, se puede elegir el modo de juego, que se clasifican en fácil, intermedio y difícil [13]. En el modo fácil, los precios se reducen un 15 %, los globos se mueven a velocidad base, se comienza con 200 vidas y se termina en la ronda 40. En el modo intermedio, el precio de las torres y las mejoras es el estándar, la velocidad de los globos aumenta un 10 %, se comienza con 150 vida y la ronda máxima es la 60. Finalmente, en el modo difícil, los costes aumentan un 8 %, los globos se mueven un 25 % más rápido, las vidas iniciales son 100 y se termina en la ronda 80 (fig. 3.11).

Cada modo de juego presenta variaciones que restringen las torres que se pueden utilizar, cambian las rondas, etc. Hay dos variaciones que hay que destacar: imposibloon y CHIMPS.

Figura 3.11: Modos de dificultad difícil

Estos dos modos se pueden considerar una dificultad por encima de difícil, ya que imposibloon aumenta los precios un 20 %, sólo se tiene una vida y se termina en la ronda 100; y CHIMPS impone varias restricciones, siendo las más relevantes disponer de una sóla vida, no poder generar dinero extra con las torres, si se pierde hay que empezar desde el inicio (en otros modos se puede pagar para continuar) y no se pueden vender las torres.

Actualizaciones del videojuego

Cada varios meses, Bloons TD6 recibe una actualización que modifica diferentes aspectos del videojuego. Dentro de estos cambios, se encuentran los "cambios de balance". Estos cambios buscan equilibrar el desempeño de las torres, mejorando aquellas que muestran un bajo rendimiento, y empeorando las que destaquen más. La forma de implementar estos cambios es mediante ajustes a los atributos de las torres, como por ejemplo el precio de sus mejoras, o la velocidad de ataque.

Entonces, si el agente aprende del videojuego en una versión concreta, en las siguientes versiones la información que ha obtenido quedará obsoleta, necesitando volver a aprender. Es por esta razón que en este proyecto se ha fijado una versión para desarrollar al agente, tal como se especifica en la restricción RES-02 en el apartado 1.2.1.

3.2. Contexto científico-técnico

En el campo de la Inteligencia Artificial, un agente inteligente es un programa diseñado para percibir un entorno, tomar decisiones en base a lo percibido, y realizar acciones que modifiquen este entorno, con el objetivo de cumplir una o varias metas [1]. Los agentes inteligentes actúan de forma racional, es decir, llevan a cabo la acción que consideran la mejor a partir de la información obtenida de la actual y anteriores percepciones. Un sistema de Inteliencia Artificial está compuesto por un agente y el entorno.

Un agente inteligente (fig. 3.12) está compuesto por sensores, para obtener el estado del entorno; actuadores, para llevar a cabo las acciones en el entorno; y el modelo, que elige qué

acción se realizará a partir del estado percibido. Para que el agente pueda aprender de sus acciones pasadas, se debe implementar en el modelo un algoritmo de Aprendizaje Automático.

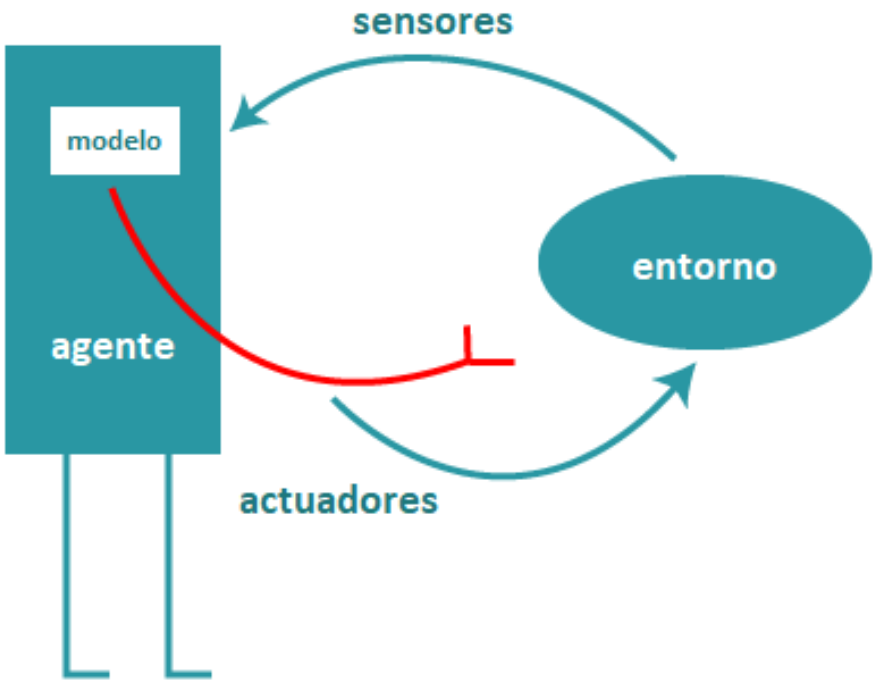


Figura 3.12: Componentes de un agente

El Aprendizaje Automático es un conjunto de técnicas que permiten al agente aprender a partir de datos. Existen cuatro ramas principales de algoritmos de Aprendizaje Automático: supervisado, en el que se tienen datos de ejemplo y los resultados que se esperan (datos etiquetados); no supervisado, donde se tienen datos de ejemplo, pero no se conocen los resultados (datos sin etiquetar); semisupervisado, si se dispone de datos etiquetados y sin etiquetar; y aprendizaje por refuerzo, en el que se aprende mediante recompensas y castigos.

Dentro del Aprendizaje Automático, se encuentra el campo del Aprendizaje Profundo, un conjunto de técnicas que imitan la forma de aprender del ser humano, permitiendo al agente encontrar patrones más complejos en los datos. Al situarse en la rama de las Redes Neuronales, se utiliza un conjunto de neuronas interconectadas entre sí formando una red, pero con más capas que las redes neuronales estándar (que es lo que permite extraer características más complejas de los datos) (fig. 3.13).

El modelo del agente inteligente de este proyecto utiliza un algoritmo de aprendizaje por refuerzo profundo. El aprendizaje por refuerzo se ajusta mejor a las características que se buscan en este agente, ya que la interacción con el videojuego se puede describir como acciones de forma intuitiva, y se recompensará o castigará al agente según las acciones que realice. Se necesita el aprendizaje profundo porque BloonsTD6 es un videojuego complejo, con muchas variables, por lo que esta herramienta facilitará el aprendizaje del agente. Esta red neuronal se encuentra en la política del agente, una función que recibe el estado actual del entorno y devuelve la acción que considera óptima para ese estado.

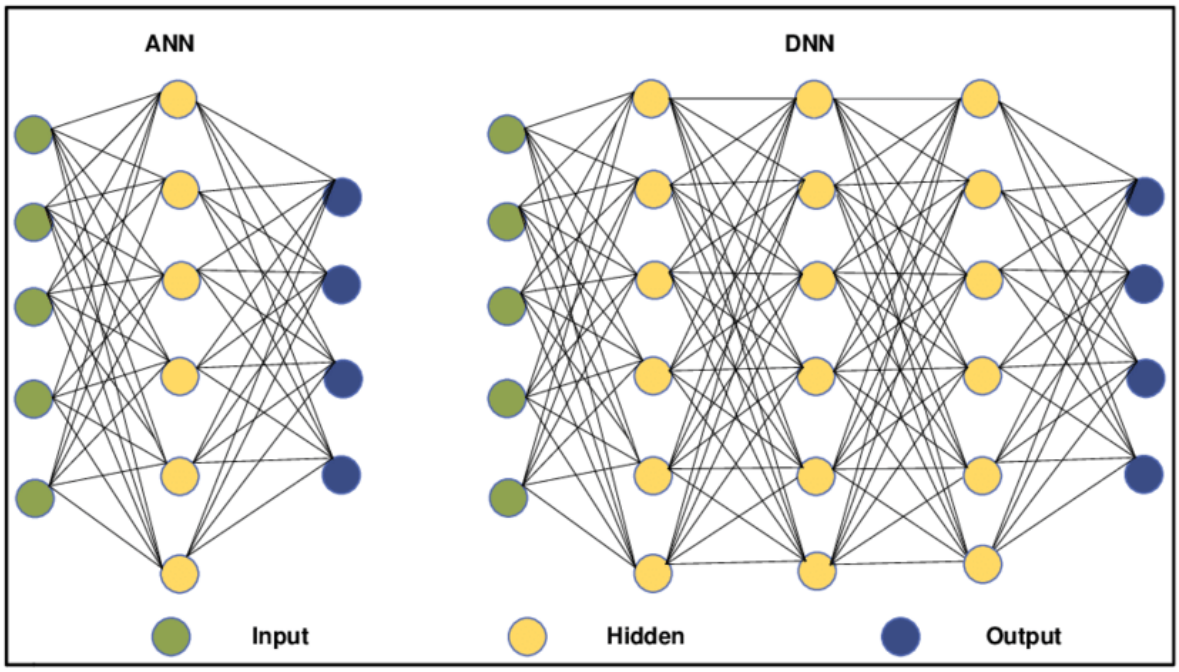


Figura 3.13: Diferencia entre una red neuronal estándar (izquierda) y una red neuronal profunda (derecha).

El algoritmo más conocido de apredizaje por refuerzo es Q-Learning, en el cual se utiliza una tabla para vincular cada par estado-acción con la recompensa que se espera que genere. Su versión con aprendizaje profundo es Deep Q-Learning, donde se sustituye la tabla por una red neuronal que recibe el estado del juego y devuelve la recompensa que se espera de realizar cada acción posible (fig. 3.14). El problema es que, si el número de acciones disponibles es muy grande o continuo, el entrenamiento del agente tardará más tiempo y requerirá más recursos (concretamente, si es continuo, es inviable).

Deep Q-Learning tiene este problema porque está orientado a predecir recompensas. Es por esto que se utiliza PPO (Optimización de la Política Proximal) [30], un algoritmo que busca optimizar la política del agente, es decir, la función que recibe el estado del entorno y devuelve la acción que considera óptima. Esta función contiene una serie de parámetros ajustables, que son lo que optimiza el agente. Durante el entrenamiento, el agente actualizará su política cada varias partidas utilizando el conjunto de estados, acciones y recompensas generados en estas.

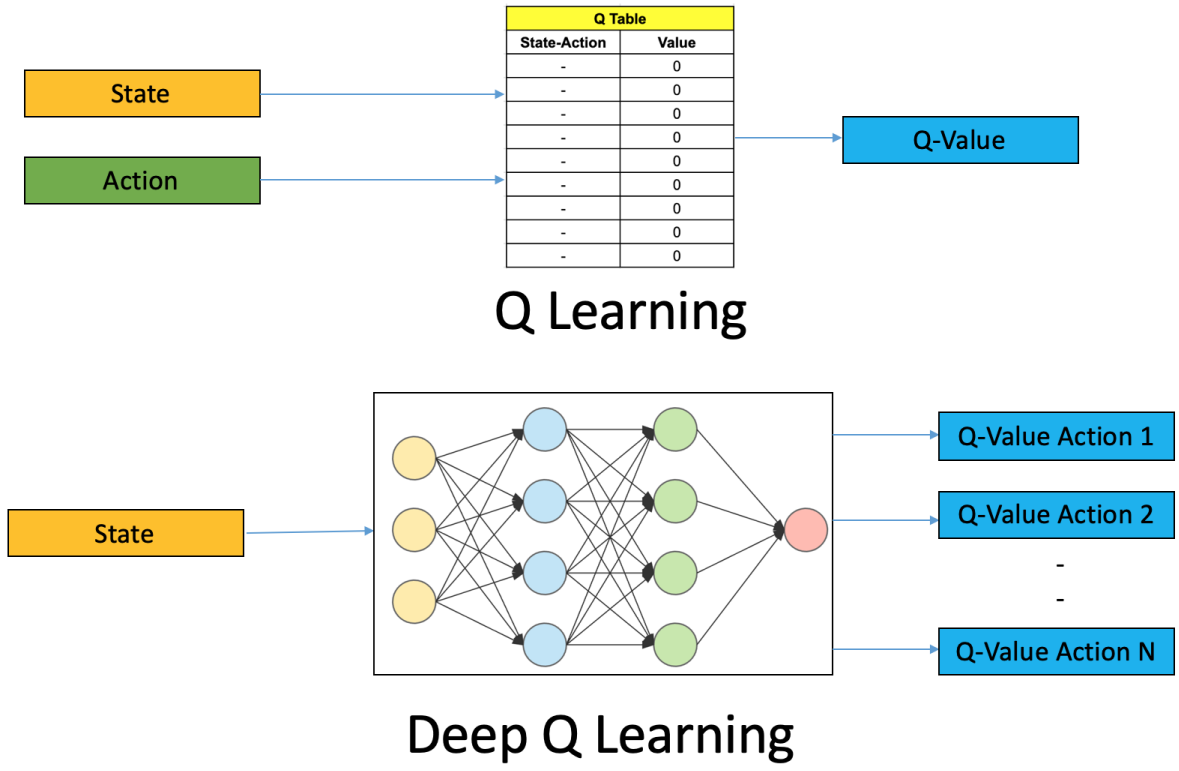


Figura 3.14: Diferencia entre Q-Learning y Deep Q-Learing

3.3. Estado del arte

3.3.1. Descripción de trabajos relacionados

Los juegos sencillos resultan fantásticos candidatos para probar los algoritmo de aprendizaje por refuerzo, ya que tienen controles y mecánicas sencillas, pero las estrategias que se pueden desarrollar son lo suficientemente profundas para evaluar estos algoritmos. Además, suelen tener incorporada una puntuación, que se puede adaptar a la recompensa total que el agente busca maximizar. En este artículo de Medium [20] se han entrenado tres agentes diferentes en tres juegos (fig. 3.15):

CartPole Consiste en mantener en equilibrio una varilla encima de un carrito. Se ha utilizado el algoritmo de aprendizaje Deep Q-Learning. El entrenamiento ha durado unos 385 segundos (50000 pasos), y tras evaluarlo en 100 partidas, obtiene una puntuación relativamente alta de forma constante.

Space Invaders El jugador controla una nave y debe acabar con los enemigos disparándolos. Se ha utilizado el algoritmo PPO, que ha sido entrenado durante 10000 pasos (no se ha indicado el tiempo). El resultado es un agente capaz de superar la mitad de la partida.

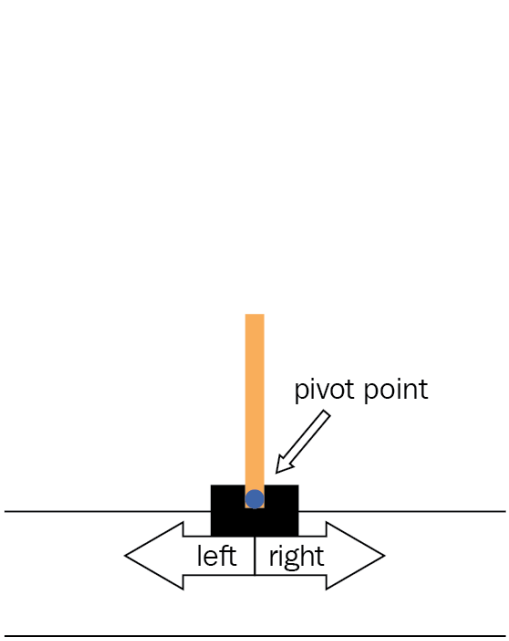
LunarLander Consiste en alunizar un módulo lunar mediante los impulsores que tiene acoplados. Para este videojuego también se utiliza PPO entrenado en 10000 pasos, y el resultado es un control pobre del módulo.

En 2017, OpenAI desarrolló "OpenAI Five", un agente capaz de jugar a Dota 2, controlando a los cinco personajes que integran un equipo. La capacidad de juego de este agente está al nivel de los jugadores profesionales, incluso ganando a los campeones del mundo [28]. Este agente también es capaz de jugar como un compañero más. El algoritmo de aprendizaje utilizado es PPO en un sistema de aprendizaje que llamado Rapid, que permite trabajar de forma simultánea cientos de GPUs y CPUs. El entrenamiento ha durado unos diez meses, durante los cuales OpenAi Five ha experimentado unos 45000 años de experiencia en Dota 2 jugando contra sí mismo.

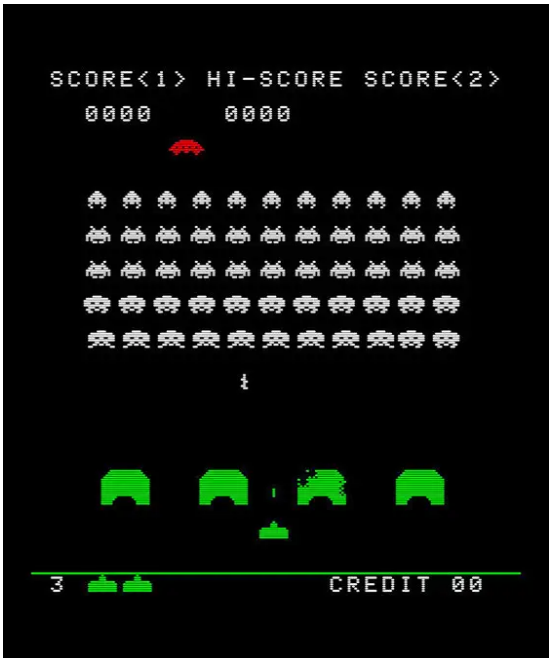
Dos años más tarde, DeepMind dio a conocer AlphaStar, un programa que juega al videojuego "Starcraft II", uno de los videojuegos de estrategia en tiempo real más complicados hasta la fecha. De forma similar a OpenAI Five, AlphaStar fue capaz de vencer a uno de los mejores jugadores profesionales de este videojuego, Grzegorz "MaNa" Komincz [3]. AlphaStar combina aprendizaje supervisado con aprendizaje por refuerzo combinando varios algoritmos propios, basados en redes Transformer, LSTM, etc. Primero se utilizó el aprendizaje supervisado para aprender de las partidas de jugadores profesionales, y tras ello, comenzó a jugar contra sí misma utilizando el aprendizaje por refuerzo. El entrenamiento duró 14 días, durante los cuales se entrenaron unos 600 agentes que experimentaron unos 200 años de experiencia de juego.

Sin embargo, hay pocos proyectos relacionados con BloonsTD6. Existen diferentes scripts que permiten completar partidas en un modo y mapa concretos, de forma automática, con el objetivo de conseguir recursos del juego. Estos scripts llevan a cabo una secuencia de acciones que aseguran superar ese modo concreto, por lo que no se les puede considerar agentes inteligentes.

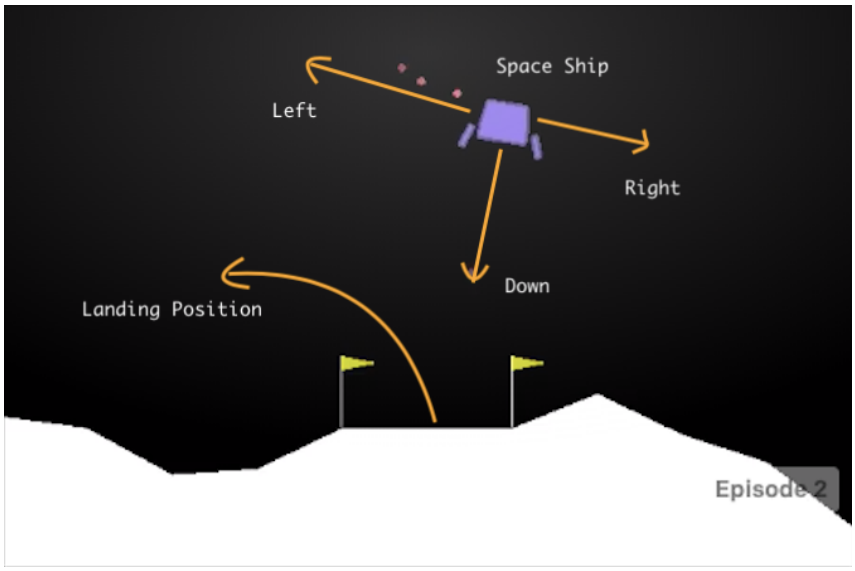
Pero hay que destacar el programa desarrollado por b2studios [5], un agente inteligente que utiliza un algoritmo genético para definir su estrategia. Este tipo de algoritmos se basan en la evolución biológica, donde las estrategias (torres que se van a colocar) son los candidatos, escogiendo las mejores y cruzándolas (combinar aspectos de las dos estrategias) para generar un



(a) CartPole



(b) Space Invaders



(c) LunarLander

Figura 3.15: Videojuegos del artículo de Medium

nuevo conjunto de estrategias. La interacción con el videojuego se lleva a cabo únicamente con el ratón, sin teclado, y el estado del videojuego se obtiene accediendo directamente a la memoria. Tras jugar durante 100 horas en el modo más difícil del juego, CHIMPS, consiguió superarlo.

3.3.2. Discusión

Los tres agentes desarrollados en el artículo de Medium permiten hacerse una idea del desempeño que otorgan los algoritmos de aprendizaje por refuerzo profundo más conocidos, Deep Q-Learning y PPO. En especial resulta interesante este segundo algoritmo, ya que es el que se utiliza en este proyecto. Los agentes que se han desarrollado han mostrado inicios de un buen comportamiento, pese a que el tiempo de entrenamiento a sido bajo. Sin embargo, sirven como una base para entender cómo va a evolucionar el agente de BloonsTD6 y la necesidad de sesiones de entrenamiento más largas.

Tanto OpenAI Five como AlphaStar son agentes mucho más grandes que el que se plantea en este proyecto, pero permiten hacerse una idea del reto que supone desarrollar un agente para un videojuego actual con la capacidad de juego de los profesionales. Requieren combinar diferentes técnicas de aprendizaje, e invertir muchos recursos para entrenarlos en el equivalente a decenas de miles de años (aunque el tiempo real de entrenamiento mejoró con creces con AlphaStar).

El agente desarrollado por B2Studios es el más relevante, al tener un objetivo similar al de este proyecto. Es un agente que utiliza un algoritmo genético, lo que implica que no tiene capacidad de aprendizaje. Esto quiere decir que la solución que encuentra sólo sirve para ese mapa y dificultad concretos, por lo que tendría que jugar otras 100 horas para superar CHIMPS en otro mapa. El agente de este proyecto debe ser capaz de transferir el conocimiento aprendido a otros mapas y modos, por ello se utiliza el aprendizaje por refuerzo. Otro factor a tener en cuenta es el tiempo de entrenamiento. El agente de B2Studios tarda 100 horas en superar CHIMPS, mientra que con PPO, en los agentes del artículo de Medium, en entrenamientos cortos ya mostraban un conocimiento básico.

Para leer el estado del videojuego, el agente de B2Studios accede a las direcciones de memoria de BloonsTD6. Para su caso, esta es una estrategia viable, pero poco escalable, ya que se necesita obtener manualmente las direcciones de memoria para cada par mapa - modo de dificultad. Además, al actualizarse el juego pueden cambiar estas direcciones, siendo necesario recuperarlas. El uso de una extensión de BloonsTD6 es mucho más escalable y robusta, al capturar directamente del juego los valores, y teniendo en cuenta que Ninja Kiwi permite las extensiones en su juego.

El agente de B2Studios utiliza únicamente el ratón para controlar el juego, teniendo que seleccionar las torres desde el inventario, iniciar la partida pulsando el botón de inicio y teniendo que mejorar las torres clicando los botones de mejora del menú de la torre, que puede aparecer a la izquierda o la derecha de la pantalla. El agente de este proyecto simplifica estos casos utilizando los atajos del teclado, ya que cada tecla corresponde a una acción, como seleccionar una torre, elegir una mejora o comenzar la partida.

Un factor a tener en cuenta es la velocidad de ejecución. En este proyecto se utiliza la extensión la extensión FasterForward, que permite acelerar el videojuego a la velocidad deseada. Gracias al uso de atajos del teclado, el agente es capaz de seguir la velocidad aumentada hasta 25 más rápida, reduciendo notablemente el tiempo de entrenamiento.

Parte II

Desarrollo de la propuesta

Capítulo 4

Descripción y desarrollo de la propuesta

El proceso de desarrollo de *software* comprende una serie de actividades sistemáticas que se llevan a cabo para crear, diseñar, implementar y mantener un programa. Al emplear la metodología ágil ASAP, el proceso de desarrollo de *software* tiene un enfoque iterativo e incremental, en el que las actividades que lo definen se realizan en cada *sprint*, entregando al final del mismo un producto funcional y de valor. Esta dinámica dota de una mayor flexibilidad y adaptabilidad al desarrollo, y permite mejorar la calidad del producto al realizar pruebas más frecuentemente.

4.1. Análisis

El análisis de *software* es el proceso de entender y definir las características y el funcionamiento del programa, así como identificar en qué condiciones se debe desarrollar. Se realiza con el objetivo de tomar decisiones informadas sobre el *software*, y representar de forma clara los requisitos que nacen de los objetivos del proyecto.

4.1.1. Requisitos de información

Los requisitos de información describen todos los aspectos relacionados con los datos creados, gestionados o emitidos por el sistema. Junto con los requisitos se incluye el diccionario de datos, un conjunto de tablas que definen los tipos de datos y sus características particulares.

RI-01 El estado del entorno se describe según los siguientes parámetros: dinero, vidas, ronda, victoria, reventones.

ID	Nombre	Definición	Tipo	Mín	Máx
A01	dinero	Moneda para comprar torres	int	0	
A02	vidas	Número de *Bloons* que se pueden escapar	int	0	5
A03	ronda	Número de oleada	int	0	10
A04	victoria	Indica si se ha superado la última ronda	bool	0	1
A05	reventones	Cantidad de globos que ha reventado cada torre	int[]	0	

Tabla 4.1: Requisito de información RI-01.

RI-02 La malla que define a los mapas es una matriz de 16 filas por 24 columnas, y los valores que pueden tomar las celdas serán: 0 para indicar que la posición está vacía; mayor que cero para representar que hay una torre; -1 para indicar que es una casilla de camino; y -2 para indicar que es un obstáculo.

RI-03 Las acciones que realiza el agente están descritas por el tipo de torre a colocar y la casilla de la matriz donde colocarla (fila y columna).

ID	Nombre	Definición	Tipo	Mín	Máx
A06	torre	Tipo de torre a colocar, siendo 0 no colocar	int	0	1
A07	fila	Número de fila en la que colocar la torre	int	0	15
A08	columna	Número de columna en la que colocar la torre	int	0	23

Tabla 4.2: Requisito de información RI-03.

RI-04 Los registros que se generan durante el entrenamiento contienen las métricas que se detallan en el apartado 5.1.1.

RI-05 Las mallas de los mapas se guardarán en la carpeta "mapas", en formato JSON. Cada malla estará almacenada en un archivo con el nombre del mapa que represente, y estará definida por una lista de listas de enteros (una matriz de enteros).

4.1.2. Casos de uso

Los casos de uso describen la secuencia de interacciones entre un actor y el sistema. Representan situaciones o escenarios en los que un usuario interactúa con el sistema para realizar una tarea o alcanzar un objetivo. Los casos de uso se describen desde la perspectiva del actor, indicando los pasos que se llevan a cabo tanto en el escenario principal (ejecución normal del caso de uso) como en escenarios alternativos (por ejemplo, si ocurre un error). Para cada caso de uso se describe también el estado en el que debe encontrarse el sistema para que pueda ejecutarse (precondiciones), así como el estado en que queda el sistema tras la ejecución (postcondiciones).

Los actores que protagonizan los casos de uso pueden ser tanto usuarios como otros sistemas *software*. En este proyecto hay tres actores: el usuario, que representa a la persona que interactúa con el programa; BTD6, que simboliza al videojuego BloonsTD6; y "Puente", que consiste en una extensión de BloonsTD6 para agregarle funcionalidades.

La figura 4.1 representa al diagrama de casos de uso que define al sistema, en el que cada actor está relacionado con los casos de uso con los que interactúa. Los casos de uso también pueden estar relacionados entre ellos: si un caso de uso precisa la ejecución de otro, se indica mediante una flecha *include*, y si añade funcionalidad a otro caso de uso, se manifiesta a través de una flecha *extend*.

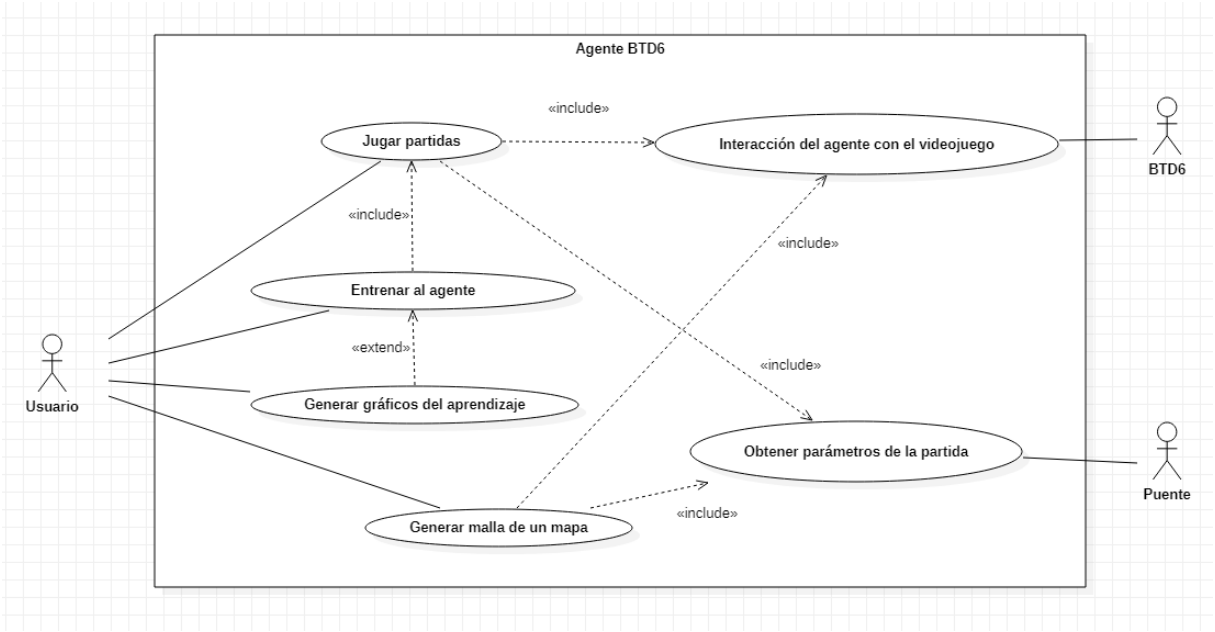


Figura 4.1: Diagrama de casos de uso

Especificación de casos de uso

UC-01	**Interacción del agente con el videojuego**
Versión	1.1
Objetivos asociados	OBJ-02.2 Construir mecanismos de acción del agente sobre el videojuego.
Requisitos asociados	
Descripción	El agente simulará el teclado y ratón para llevar a cabo acciones como colocar torres o reiniciar una partida.
Precondición	El usuario debe haber iniciado el juego.
Secuencia normal	1. El agente recibe la acción que se realizará. 2. El agente utiliza atajos del teclado para seleccionar un tipo de torre o comenzar la partida. 3. El agente simula un *click* en las coordenadas donde se colocará la torre.
Postcondición	Se ha reiniciado la partida, o se ha colocado una torre en la posición correspondiente.
Excepciones	Si la acción recibida consiste en colocar una torre donde no esté permitido, ya sea porque el mapa no lo permite o ya existe otra torre en esa posición, no se llevará a cabo esa misma acción.

Tabla 4.3: Caso de uso UC-01: Interacción del agente con el videojuego.

UC-02	**Obtener parámetros de la partida**
Versión	1.1
Objetivos asociados	OBJ-02.1 Obtención del estado del videojuego por parte del agente.
Requisitos asociados	
Descripción	El agente se comunica con la extensión de BloonsTD6 (denominada "Puente") para obtener el estado actual de la partida.
Precondición	Se ha abierto en el videojuego uno de los mapas.
Secuencia normal	1. La extensión Puente abre una tubería nombrada para iniciar la conexión. 2. El agente se conecta a la tubería nombrada. 3. El agente solicita los parámetros a la extensión Puente. 4. La extensión puente lee el estado de la partida y envía los parámetros al agente. 5. Se repiten los pasos 3 y 4 hasta que termine la sesión de juego. 6. Se cierra la tubería nombrada, terminando la conexión.
Postcondición	La extensión Puente queda a la espera de una nueva conexión con el agente.
Excepciones	Si se produce algún error durante la comunicación, se muestra la causa del fallo y se avanza al último paso.

Tabla 4.4: Caso de uso UC-02: Obtener parámetros de la partida.

UC-03	**Jugar partidas**
Versión	1.1
Objetivos asociados	OBJ-03 Desarrollar el agente.
Requisitos asociados	▪ UC-01 Interacción del agente con el videojuego ▪ UC-02 Obtener parámetros de la partida
Descripción	El agente jugará un número concreto de partidas en el mapa indicado, y le proporcionará al usuario estadísticas sobre la sesión de juego.
Precondición	El usuario debe haber iniciado el juego.
Secuencia normal	1. El usuario selecciona y se sitúa en el mapa en el que se realizará la sesión de juego. 2. El usuario carga al agente, e inicia la sesión de juego indicando el mapa y el número de partidas. 3. El agente juega el número indicado de partidas y devuelve estadísticas sobre la sesión de juego.
Postcondición	El videojuego se encuentra al inicio de una partida.
Excepciones	Si el número de partidas es menor a uno o el mapa que se le indica al agente no existe o no está disponible, no se llevará a cabo la sesión de juego.

Tabla 4.5: Caso de uso UC-03: Jugar partidas.

UC-04	**Entrenar al agente**
Versión	1.0
Objetivos asociados	OBJ-03.2 Entrenar al agente
Requisitos asociados	UC-03 Jugar partidas
Descripción	El agente juega episodios hasta alcanzar un límite máximo de pasos. Tras ejecutar una cantidad definida de pasos, se actualizará la política de su modelo y se genera un registro con las métricas de esa actualización.
Precondición	Se ha abierto en el videojuego uno de los mapas.
Secuencia normal	1. El usuario establece los parámetros del aprendizaje, e inicia el entrenamiento. 2. El agente juega episodios durante una cantidad de pasos establecida. 3. El agente utiliza los episodios desde la última actualización para actualizar su política y añadir al registro las métricas obtenidas. 4. Se repiten los pasos 2 y 3 hasta alcanzar el número máximo de pasos establecido por el usuario. 5. Se guarda al agente entrenado.
Postcondición	El videojuego se encuentra en medio de la última partida que estaba jugando el agente. Se ha guardado el agente entrenado y el registro con las métricas.
Excepciones	Si se produce algún error que impida el entrenamiento, se muestra la causa del fallo, se detiene el entrenamiento y se avanza al último paso.

Tabla 4.6: Caso de uso UC-04: Entrenar al agente.

UC-05	**Generar gráficos del aprendizaje**
Versión	1.0
Objetivos asociados	OBJ-03.2 Entrenar al agente
Requisitos asociados	UC-04 Entrenar al agente
Descripción	El sistema genera gráficos sobre las métricas obtenidas en el entrenamiento de un agente que el usuario puede consultar.
Precondición	Se ha entrenado a un agente y se dispone del registro del entrenamiento.
Secuencia normal	1. El usuario introduce la ruta del registro del entrenamiento. 2. El sistema genera los gráficos sobre las métricas de ese entrenamiento, y los guarda en una imagen.
Postcondición	Se muestran los gráficos generados, y se ha guardado una imagen con los mismos.
Excepciones	Si durante el entrenamiento sólo se ha producido una actualización, no se generan los gráficos.

Tabla 4.7: Caso de uso UC-05: Generar gráficos del aprendizaje.

UC-06	**Generar malla de un mapa**
Versión	1.1
Objetivos asociados	OBJ-01 Describir el videojuego como el entorno del agente.
Requisitos asociados	▪ UC-01 Interacción del agente con el videojuego ▪ UC-02 Obtener parámetros de la partida
Descripción	El sistema coloca torres en todas las posiciones de un mapa para generar una matriz en la que se indique en qué posiciones se pueden colocar torres y en cuáles no.
Precondición	El usuario debe haber iniciado el juego.
Secuencia normal	1. El usuario selecciona y se sitúa en el mapa del cual se generará la malla. 2. El usuario inicia el proceso de creación de la malla. 3. El sistema coloca torres en todas las posiciones posibles, y marca en la casilla correspondiente de la matriz si es una posición permitida o no. 4. El sistema guarda la malla generada en el fichero de mapas.
Postcondición	Se ha agregado la malla del nuevo mapa en el fichero de mapas.
Excepciones	Si existe ya la malla del mapa indicado, no se genera una nueva. Si el mapa presenta paredes, el usuario debe modificar manualmente la matriz indicando en qué casillas hay paredes.

Tabla 4.8: Caso de uso UC-06: Generar malla de un mapa.

4.1.3. Requisitos funcionales

Los requisitos funcionales describen las capacidades y funciones que debe proveer el sistema *software*, es decir, definen cómo se debe comportar el sistema y qué funcionalidades debe ofrecer. Estos requisitos deben proporcionar una descripción lo suficientemente detallada y sin ambigüedad para permitir el desarrollo del sistema. Los requisitos funcionales se obtienen al especificar las funcionalidades que conforman los escenearios que describen los casos de uso; por ello, estos requisitos se organizan según los casos de uso.

UC-01: Interacción del agente con el videojuego

RF-01 El agente debe ser capaz de colocar una tipo de torre en la fila y columna que reciba como acción, si hay dinero disponible para comprarla y no hay ningún obstáculo u otra torre en esa casilla. Se almacenará el número asociado al tipo de torre en la posición de la matriz.

RF-02 El agente simulará el teclado para seleccionar los tipos de torre (letras), comenzar la partida (espacio) y acelerar y ralentizar la ejecución (teclas f1 y f4).

RF-03 El agente transformará en coordenadas en píxeles la fila y columna que reciba en la acción, y clicará en esas coordenadas para colocar una torre.

UC-02: Obtener parámetros de la partida

RF-04 La extensión Puente debe crear una tubería nombrada antes de comenzar la partida, y el agente debe conectarse a ella.

RF-05 La extensión Puente debe enviar al agente a través de la tubería nombrada los parámetros del estado de la partida descritos en el RI-01, en formato JSON, cuando reciba una petición de parámetros.

RF-06 La extensión Puente debe vaciar la tubería nombrada y eliminarla al terminar la conexión.

UC-03: Jugar partidas

RF-07 El agente debe ser capaz de cargar un modelo guardado en la ruta que indique el usuario.

RF-08 El usuario debe escoger el mapa donde jugará el agente, el número de partidas y el modelo que utilizará.

RF-09 El agente devolverá la media y desviación típica de las recompensas obtenidas en los episodios jugados.

UC-04: Entrenar al agente

RF-10 El usuario debe establecer los parámetros del aprendizaje: tasa de aprendizaje, pasos por actualización y número de actualizaciones.

RF-11 El agente debe generar un registro con los valores de las métricas obtenidas al actualizar la política en una carpeta con la hora del inicio del entrenamiento. El archivo se llamará "logs".

RF-12 El agente debe guardar una copia de su modelo al finalizar el entrenamiento, con el nombre "ppo_btd6".

UC-05: Generar gráficos del aprendizaje

RF-13 El agente debe mostrar y guardar en una imagen "graficos.png" los gráficos de las métricas del entrenamiento, utilizando el registro generado en el mismo. Las métricas serán las siguientes: recompensa media, duración media, varianza explicada, pérdida y entropía, utilizando como abcisas el número de iteraciones.

UC-06: Generar malla de un mapa

RF-14 El sistema debe ser capaz de generar una matriz de posiciones permitidas, según el RI-02. Debe agregar la matriz al fichero "Mapas.json" con el nombre del mapa.

RF-15 El usuario deberá introducir manualmente las posiciones de la matriz que representan un obstáculo.

4.1.4. Requisitos no funcionales

Los requisitos no funcionales representan características generales o restricciones del sistema *software*. A diferencia de los requisitos funcionales, no definen qué funcionalidades debe tener el sistema, sino que describen cómo debe ser este. Existe una amplia variedad de requisitos no funcionales, dependiendo del aspecto del sistema que describen, como pueden ser la usabilidad, la eficiencia, la seguridad y la fiabilidad.

Requisitos de usabilidad

Los requisitos de usabilidad se centran en mejorar la experiencia del usuario al utilizar el sistema *software*.

RNF-01	**Mostrar métricas durante el entrenamiento**
Requisitos asociados	UC-04: Entrenar al agente
Descripción	El sistema debe mostrar en cada actualización una tabla con las métricas de hasta el momento.

Tabla 4.9: Requisito no funcional RNF-01: Mostrar métricas durante el entrenamiento.

RNF-02	**Mostrar barra de progreso**
Requisitos asociados	UC-04: Entrenar al agente
Descripción	El sistema debe mostrar una barra de progreso según los pasos realizados. También se mostrará el tiempo restante estimado según los pasos por segundo.

Tabla 4.10: Requisito no funcional RNF-02: Mostrar barra de progreso.

RNF-03	**Crear carpetas de registro y modelos**
Requisitos asociados	UC-04: Entrenar al agente
Descripción	El sistema creará las carpetas para el registro y los modelos de forma automática si no existiesen.

Tabla 4.11: Requisito no funcional RNF-03: Crear carpetas de registro y modelos.

RNF-04	**Mostrar carpetas de registro y modelos**
Requisitos asociados	UC-04: Entrenar al agente
Descripción	El sistema mostrará al inicio del entrenamiento las rutas donde se guardarán el registro del entrenamiento y los modelos generados tras cada actualización.

Tabla 4.12: Requisito no funcional RNF-04: Mostrar carpetas de registro y modelos.

RNF-05	**Pausa al jugar**
Requisitos asociados	UC-03: Jugar partidas
Descripción	El usuario podrá pausar el entrenamiento al llevar el cursor a la esquina superior izquierda. Para retomar el aprendizaje, el usuario deberá colocar el cursor en la esquina inferior izquierda.

Tabla 4.13: Requisito no funcional RNF-05: Pausa al jugar.

Requisitos de eficiencia

Los requisitos de eficiencia buscan mejorar el rendimiento y el uso eficiente de los recursos al utilizar el programa.

RNF-06	**Tiempos de espera**
Requisitos asociados	UC-03: Jugar partidas
Descripción	El agente dejará un tiempo de espera de 0.05 segundos entre pasos para reducir la carga de procesamiento y evitar información redundante.

Tabla 4.14: Requisito no funcional RNF-06: Tiempos de espera.

RNF-07	**Velocidad de partida**
Requisitos asociados	UC-03: Jugar partidas
Descripción	El juego se acelerará x25 al colocar la primera torre de la partida mediante la extensión *fasterforward*, para reducir el tiempo de entrenamiento.

Tabla 4.15: Requisito no funcional RNF-07: Velocidad de partida.

Requisitos de fiabilidad

Los requisitos de fiabilidad se encargan de asegurar que el sistema *software* pueda mantener un nivel de funcionamiento y desempeño.

RNF-08	**Generar copias del modelo durante el entrenamiento**
Requisitos asociados	UC-04: Entrenar al agente
Descripción	El agente guardará una copia del modelo cada vez que actualice su política, en una carpeta con la hora del inicio del entrenamiento, y agregando el número de pasos totales hasta el momento.

Tabla 4.16: Requisito no funcional RNF-08: Generar copias del modelo durante el entrenamiento.

RNF-09	**Iniciar el entrenamiento desde una copia del modelo**
Requisitos asociados	UC-04: Entrenar al agente
Descripción	El agente podrá inicializar su modelo a partir de una copia generada durante el entrenamiento o al terminar este.

Tabla 4.17: Requisito no funcional RNF-09: Iniciar el entrenamiento desde una copia del modelo.

Requisitos de seguridad

Los requisitos de seguridad se centran en la protección tanto del sistema y como de la información con la que trabaja.

RNF-10	**Cerrar la tubería nombrada al terminar la conexión**
Requisitos asociados	UC-02: Obtener parámetros de la partida
Descripción	Tanto el agente como la extensión Puente deben asegurarse de vaciar y cerrar la tubería nombrada cuando se termine la conexión.

Tabla 4.18: Requisito no funcional RNF-10: Cerrar la tubería nombrada al terminar la conexión.

RNF-11	**Extensión de sólo lectura**
Requisitos asociados	UC-02: Obtener parámetros de la partida
Descripción	La extensión Puente no podrá modificar valores del videojuego, sólo leerlos, para evitar una posible suspensión de la cuenta.

Tabla 4.19: Requisito no funcional RNF-11: Extensión de sólo lectura.

4.2. Diseño

El diseño de *software* consiste en la creación de una representación conceptual y técnica siguiendo la especificación del programa obtenida en el análisis. En la etapa de diseño se define la estructura, arquitectura y los componentes, estableciendo la base para la construcción del sistema. Para llevar a cabo el diseño se utilizan diferentes diagramas, para facilitar la comprensión del sistema de una forma sencilla y visual.

4.2.1. Arquitectura física

El diagrama de arquitectura física muestra cómo se estructuran los componentes físicos que forman parte del sistema *software* o que interactúan con él. Gracias a este diagrama se puede comprender de forma sencilla cómo se relacionan los componentes físicos y cómo están distribuidos en la infraestructura del sistema. En la figura 4.2 se muestra el diagrama de arquitectura física del proyecto, compuesto esencialmente por el ordenador en el cual se ejecuta tanto el agente inteligente como el videojuego.

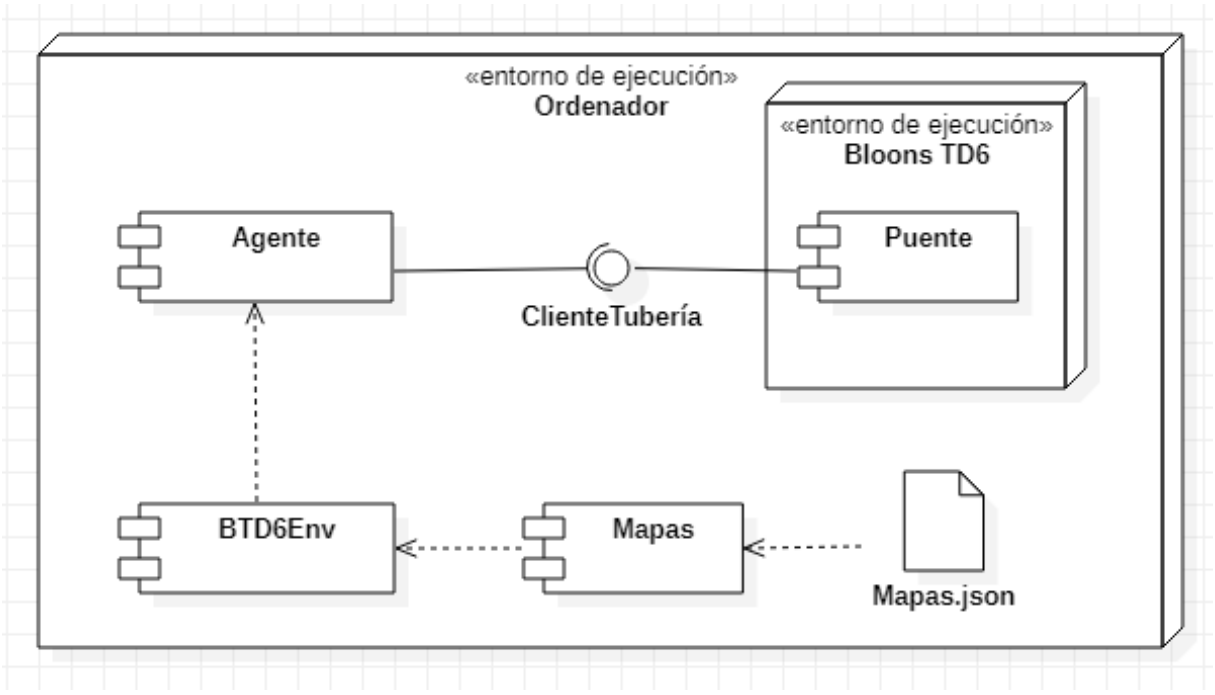


Figura 4.2: Diagrama de arquitectura física

4.2.2. Arquitectura lógica

El diagrama de arquitectura lógica muestra la organización estructural del sistema desde un punto de vista conceptual y funcional. De igual forma que el diagrama de arquitectura física representa la distribución y relaciones de los componentes físicos, el diagrama de arquitectura lógica describe cómo se relacionan los componentes lógicos y en qué capas se distribuyen. En la figura 4.3 se muestra el diagrama de arquitectura lógica del proyecto, compuesto principalmente por dos capas: la capa de lógica, en la que se encuentran los módulos correspondientes al agente y al entorno; y la capa de acceso a datos, que contiene los módulos para generar y leer las mallas de los mapas, y el cliente de la conexión con la extensión Puente.

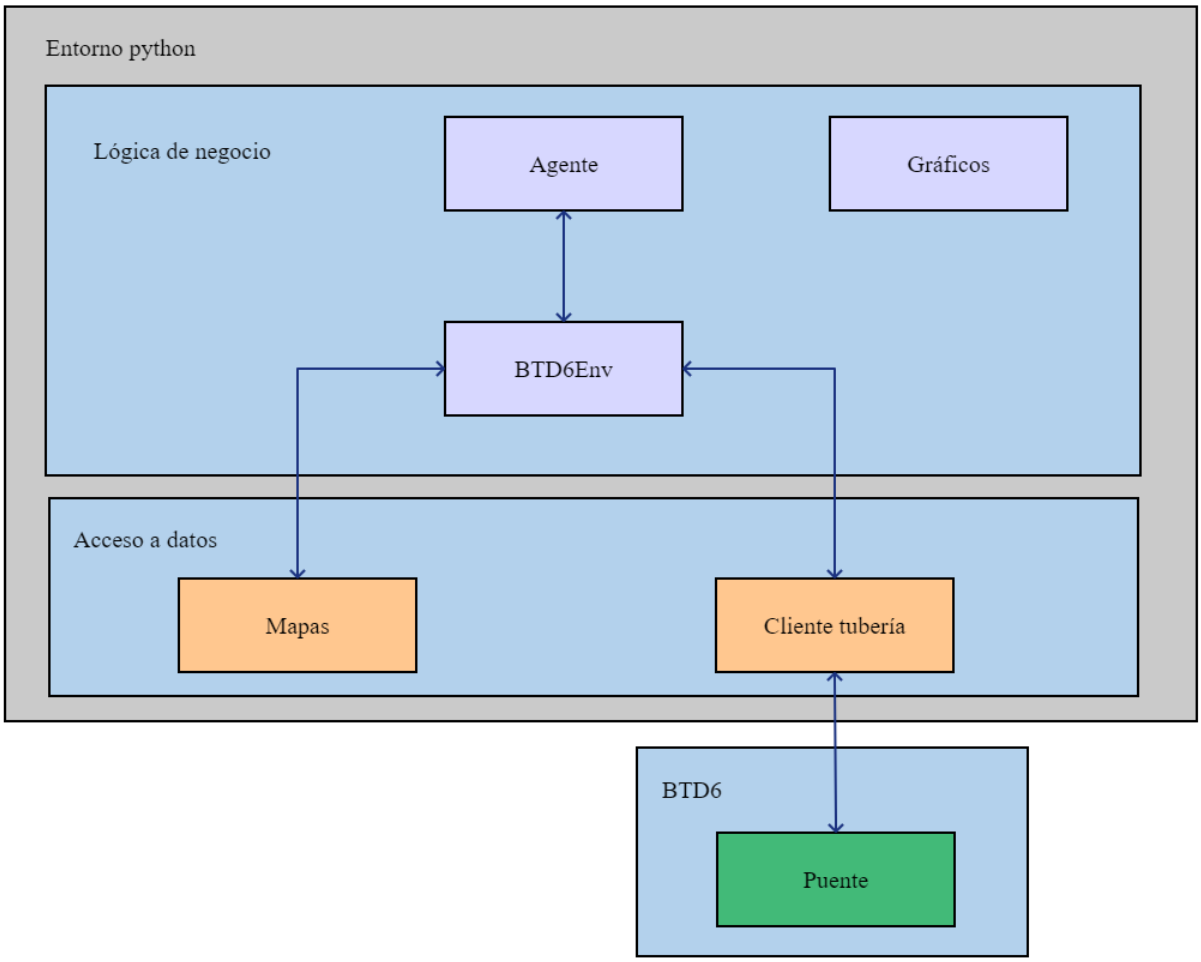


Figura 4.3: Diagrama de arquitectura lógica

4.2.3. Diagrama de estados

El diagrama de estados describe el comportamiento del agente frente a eventos, modelando su ciclo de vida. Está compuesto por estados, durante los cuales el agente está realizando una actividad, y transicciona a otro estado cuando ocurre un evento (termina una actividad, recibe un mensaje, etc). Se ha utilizado este diagrama para representar el ciclo de vida del agente

durante una partida. El diagrama comienza en el estado inicial, representado por un círculo negro, que se corresponde con la llamada al método "jugar" del agente; y termina en el estado final, representado por un círculo con una circunferencia alrededor.

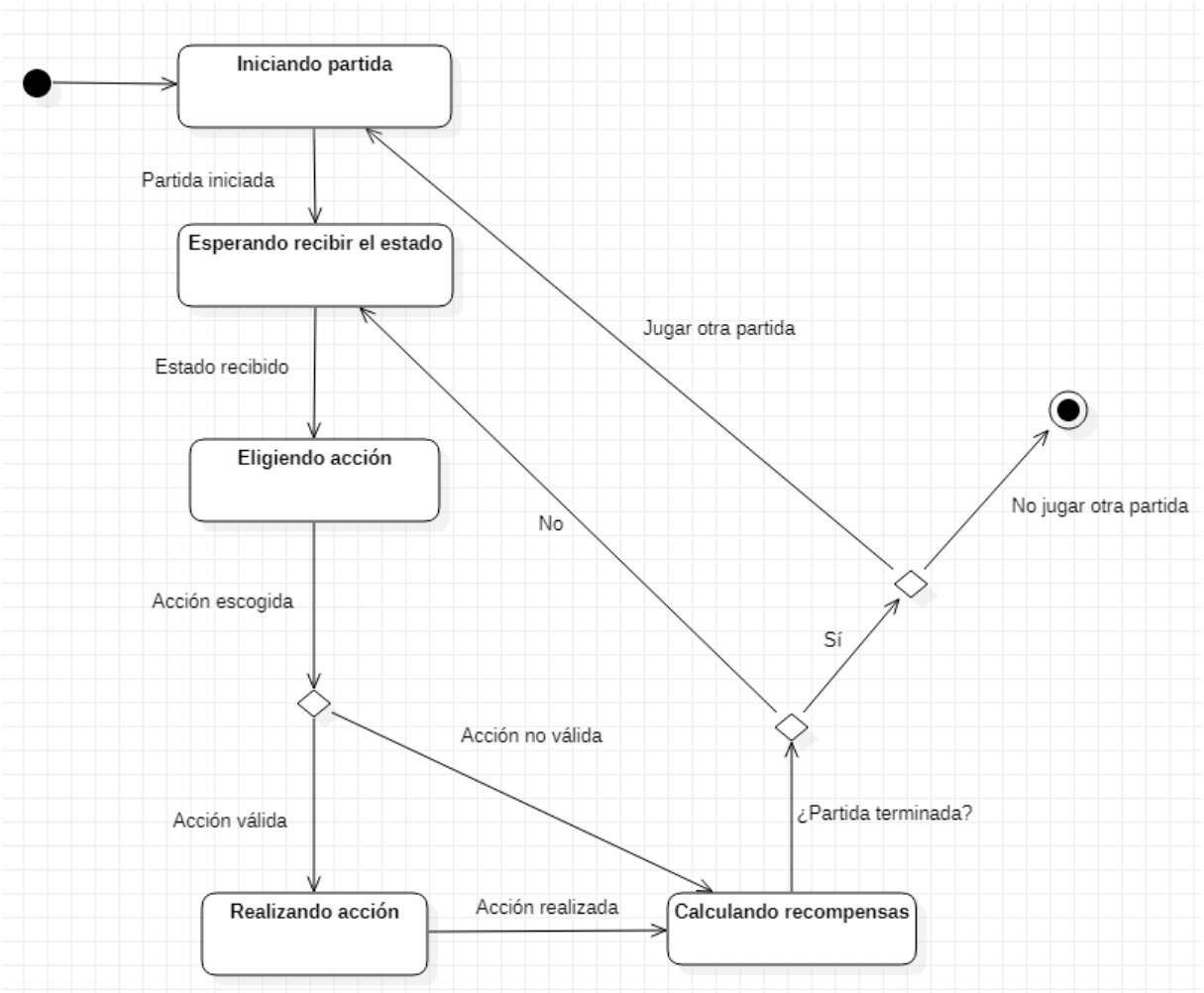


Figura 4.4: Diagrama de estados

4.2.4. Comunicación del agente con el Puente

El caso de uso UC-02 "Obtener parámetros de la partida" describe cómo se comunica el agente con la extensión Puente para obtener el estado de la partida. Para facilitar la comprensión de esta interacción, la figura 4.5 muestra el diagrama de comunicación correspondiente. Este tipo de diagrama describe la interacción entre varios componentes del sistema mediante el paso de mensajes. Para ello, se representa la línea de vida de cada componente en vertical, y los mensajes que se envían y reciben se representan en horizontal.

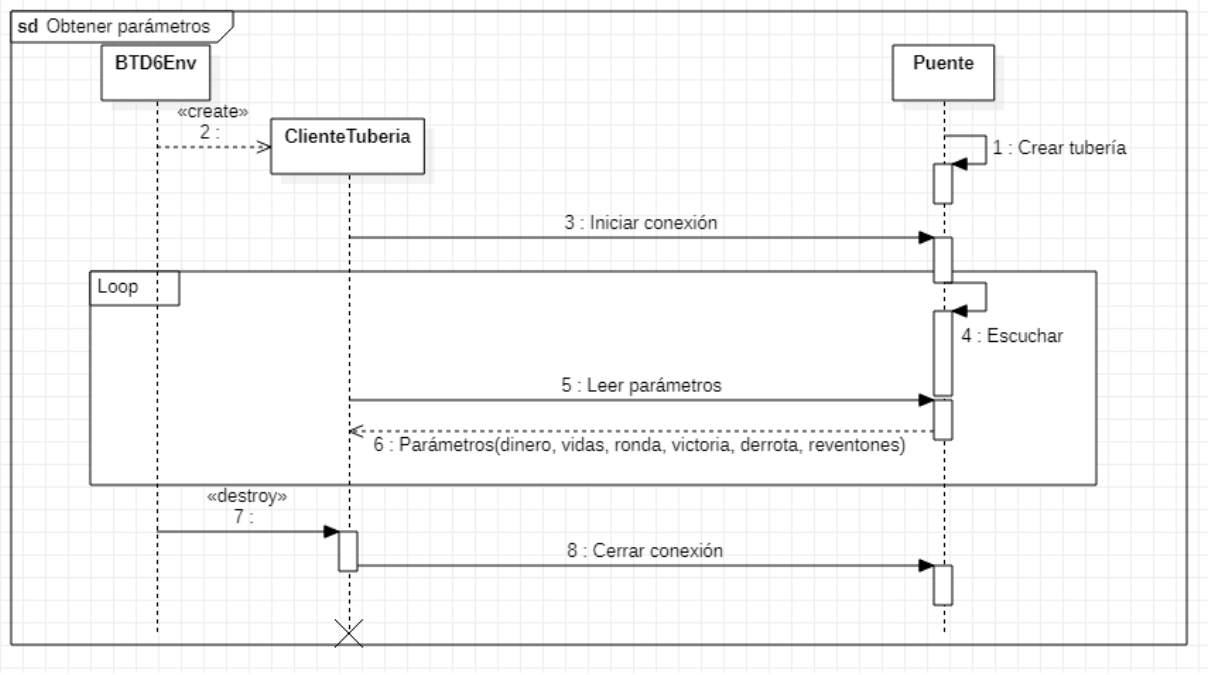


Figura 4.5: Diagrama de comunicación para el caso de uso CU-02

4.2.5. Diagrama de clases

El diagrama de clases (fig. 4.6) es una representación gráfica de las clases que conforman al sistema. Cada clase está compuesta por atributos y métodos, que describen sus propiedades y cómo interactúa con su entorno, respectivamente. El diagrama de clases también define las relaciones entre las clases, que se dividen en tres categorías: dependencia (una clase hace uso de otra), composición y agregación (una clase está compuesta por otras) y herencia (una clase hereda el comportamiento de otra).

4.3. Implementación

En la etapa de implementación se construye el programa, convirtiendo la especificación y diseño del *software* en código ejecutable y de esta forma constituir el producto funcional. Esta fase incluye el entrenamiento y ajuste del agente inteligente.

En cada *sprint* se ha desarrollado un conjunto de funcionalidades del producto, hasta tener una versión básica pero funcional del agente inteligente y su entorno. Se comenzó a implementar los actuadores del agente, es decir, su capacidad de colocar torres, utilizando el módulo de Python *pydirectinput* [21], el cual permite simular el teclado y el ratón.

La siguiente parte fue desarrollar los sensores del agente. Al principio, se optó por leer los valores de la memoria del juego a través de sus direcciones de memoria. Este método requería obtener las direcciones de memoria manualmente, utilizando el programa *Cheat Engine* [11] el cual permite buscar y filtrar direcciones de memoria mediante ingeniería inversa (fig. 4.7). Tras obtener las direcciones estáticas, es decir, que no cambian entre partidas, se hacía uso del módulo *pymem* [34] para leer los valores durante la ejecución.

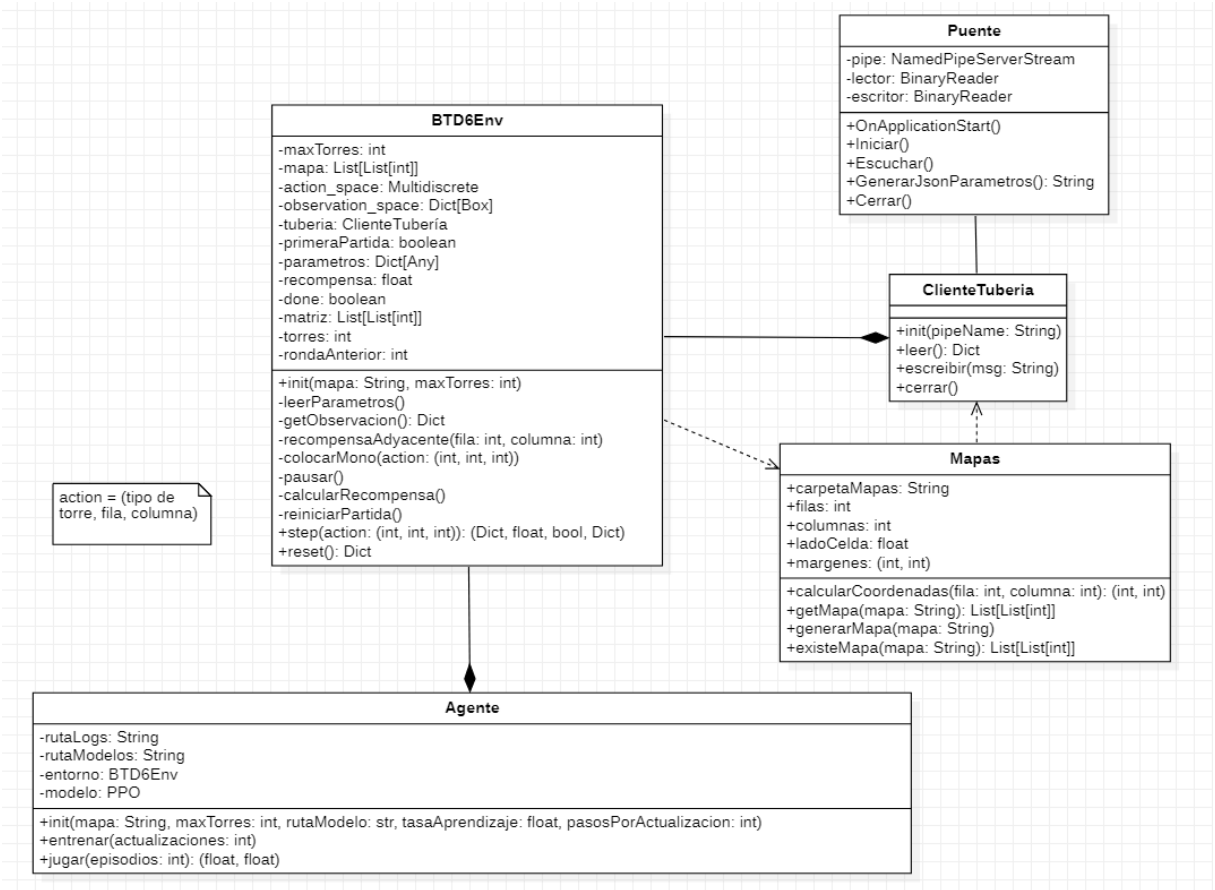


Figura 4.6: Diagrama de clases

Sin embargo, este método tenía un grave problema: las actualizaciones del juego pueden alterar las direcciones estáticas a la memoria, lo que provoca que se tengan que volver a obtener. Además, para cada modo de juego existen direcciones diferentes, aumentando así la carga de trabajo.

Por estas razones se optó por el método actual: crear una extensión del videojuego que se encargue de obtener el estado de la partida y transmitírselo al agente. Esta estrategia es por un lado mucho más robusta, ya que Ninja Kiwi permite el uso de extensiones en BloonsTD6, y por otro más sencilla y escalable, al poder acceder a todos los parámetros del juego en cualquier momento.

El siguiente paso es traducir el videojuego en el entorno con el que interactúa el agente. La estructura del entorno sigue las pautas descritas en *stable-baselines3* [35] al implementar la interfaz Gym. Esta interfaz exige definir el espacio de observaciones (*observation_ space*), es decir, el estado de la partida; y el espacio de acciones (*action_ space*). Además, es necesario definir dos métodos: *step* y *reset*. El primero, *step*, contiene la mayor parte de la lógica del entorno: se encarga de aplicar la acción que recibe como parámetro, computa el estado en que queda el entorno tras esa acción y calcula las recompensas. El segundo, *reset*, se encarga de inicializar el estado del entorno y comenzar una nueva partida.

Con el entorno definido, se puede crear un modelo PPO. Por eficiencia y compatibilidad,

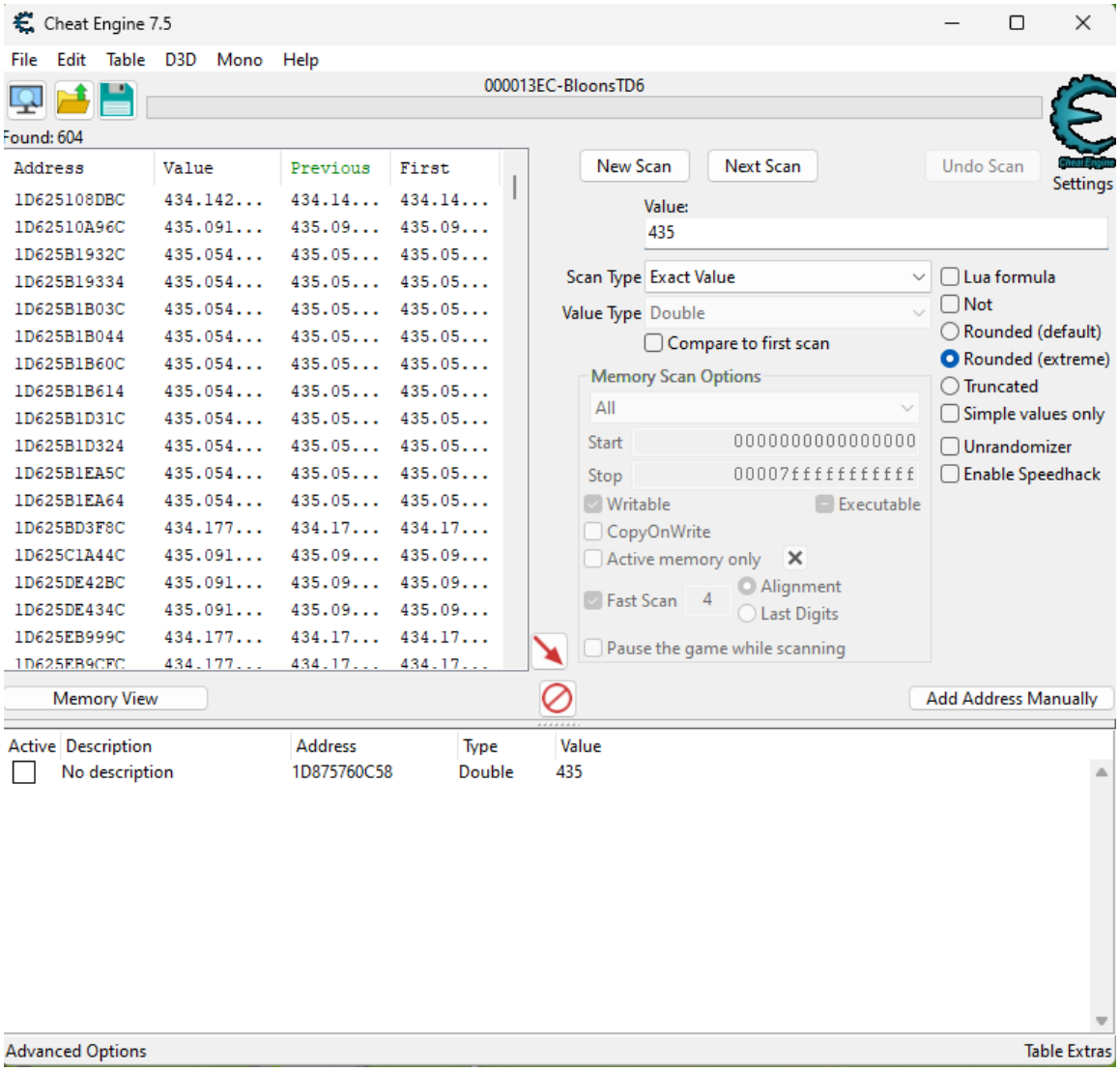


Figura 4.7: Interfaz de Cheat Engine. Búsqueda del puntero del dinero.

hay que reducir la dimensionalidad de los espacios de observaciones y acciones 4.8. Para ello se envuelve el entorno con el envoltorio *FlattenObservation*, el cual se encargará de transformar ambos espacios en vectores de una dimensión. Junto con este se agrega el envoltorio *Monitor* y un *logger*, los cuales se encargarán de generar el registro de métricas en el entrenamiento. Finalmente, se agrega la barra de progreso y la generación de copias del modelo tras cada actualización, ambas funcionalidades que ofrece *stable-baselines3*. Con esto el modelo está completo.

La última parte es entrenar y refinar al agente. Para ello, se le indica cuántos pasos se desea que dure el entrenamiento, y comenzará a jugar partidas hasta que llegue a esta cantidad. Una vez terminado el aprendizaje, se guarda el modelo entrenado en un archivo zip. Además, se puede utilizar el registro generado para ver los gráficos de las métricas y evaluar el desempeño del agente.

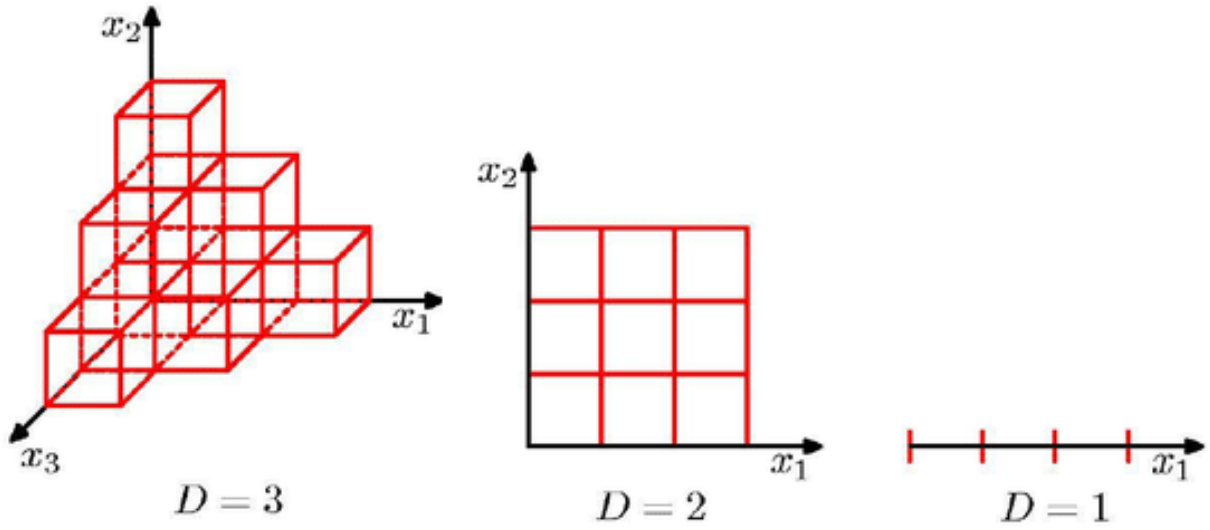


Figura 4.8: Representación de reducción de dimensionalidad. Orígen: feedingthemachine.ai

4.4. Pruebas

Las pruebas representan una revisión final de las especificaciones, el diseño y la implementación. Una prueba es el proceso de ejecutar el programa con el objetivo de encontrar errores. Durante la implementación se llevan a cabo pruebas locales sobre el código que se esté desarrollando, pero es necesario diseñar un modelo de pruebas que permita una validación del sistema *software* más completa.

La evaluación de este sistema se realizará utilizando pruebas de caja negra. Este tipo de pruebas se centran en la función específica para la cual fue creado cada componente, es decir, comprueban qué hace el sistema, no cómo lo hace. Cada prueba se compone de una descripción, unas precondiciones, las entradas proporcionadas, la salida obtenida y la salida esperada.

CN-01	**Conexión con la extensión Puente**
Descripción	Comprueba que la transmisión de los parámetros del estado de la partida se realiza correctamente.
Precondición	El videojuego debe estar iniciado, y se ha entrado en el mapa “troncos” en la dificultad “fácil-estándar”
Datos de entrada	Se ejecuta la petición del estado.
Resultado esperado	Un diccionario con los parámetros: dinero = 650, vidas = 200, ronda = 1, victoria = False, reventones = []
Resultado obtenido	Un diccionario con los parámetros: dinero = 650, vidas = 200, ronda = 1, victoria = False, reventones = []

Tabla 4.20: Prueba de caja negra CN-01: Conexión con la extensión Puente.

CN-02	**Parámetros de Agente: mapa**
Descripción	Comprueba que no se pueda crear un agente al introducir un mapa que no ha sido generado.
Precondición	El videojuego debe estar iniciado. No debe existir el mapa “aaa”.
Datos de entrada	Agente(mapa = “aaa”, maxTorres = 2)
Resultado esperado	Se lanza una excepción RuntimeError indicando que no se ha generado el mapa “aaa”.
Resultado obtenido	Se lanza una excepción RuntimeError indicando que no se ha generado el mapa “aaa”.

Tabla 4.21: Prueba de caja negra CN-02: Parámetros de Agente: mapa.

CN-03	**Parámetros de Agente: torres máximas**
Descripción	Comprueba que no se pueda crear un agente al introducir un número de torres máximas menor que 1.
Precondición	El videojuego debe estar iniciado. Debe existir el mapa “troncos”.
Datos de entrada	Agente(mapa = “troncos”, maxTorres = 0)
Resultado esperado	Se lanza una excepción ValueError indicando que el parámetro “maxTorres” debe ser mayor que 0.
Resultado obtenido	Se lanza una excepción ValueError indicando que el parámetro “maxTorres” debe ser mayor que 0.

Tabla 4.22: Prueba de caja negra CN-03: Parámetros de Agente: torres máximas.

CN-04	**Parámetros de Agente: ruta del modelo**
Descripción	Comprueba que no se pueda crear un agente al introducir una ruta de un archivo del modelo que no existe.
Precondición	El videojuego debe estar iniciado. No debe existir un modelo llamado “ruta”.
Datos de entrada	Agente(mapa = “troncos”, maxTorres = 2, rutaModelo = “ruta”)
Resultado esperado	Se lanza una excepción FileNotFoundError indicando que no existe el fichero “ruta.zip”.
Resultado obtenido	Se lanza una excepción FileNotFoundError indicando que no existe el fichero “ruta.zip”.

Tabla 4.23: Prueba de caja negra CN-04: Parámetros de Agente: ruta del modelo.

CN-05	**Parámetros de Agente: tasa de aprendizaje**
Descripción	Comprueba que no se pueda crear un agente al introducir una tasa de aprendizaje igual o menor que 0.
Precondición	El videojuego debe estar iniciado. Debe existir el mapa "troncos".
Datos de entrada	Agente(mapa = "troncos", maxTorres = 2, tasaAprendizaje = 0)
Resultado esperado	Se lanza una excepción ValueError indicando que el parámetro "tasaAprendizaje" debe ser mayor que 0.
Resultado obtenido	Se lanza una excepción ValueError indicando que el parámetro "tasaAprendizaje" debe ser mayor que 0.

Tabla 4.24: Prueba de caja negra CN-05: Parámetros de Agente: tasa de aprendizaje.

CN-06	**Parámetros de Agente: pasos por actualización**
Descripción	Comprueba que no se pueda crear un agente al introducir un número de pasos por actualización igual o menor que 0.
Precondición	El videojuego debe estar iniciado. Debe existir el mapa "troncos".
Datos de entrada	Agente(mapa = "troncos", maxTorres = 2, pasosPorActualizacion = 0)
Resultado esperado	Se lanza una excepción ValueError indicando que el parámetro "pasosPorActualizacion" debe ser mayor que 0.
Resultado obtenido	Se lanza una excepción ValueError indicando que el parámetro "pasosPorActualizacion" debe ser mayor que 0.

Tabla 4.25: Prueba de caja negra CN-06: Parámetros de Agente: pasos por actualización.

CN-07	**Número de actualizaciones del entrenamiento**
Descripción	Comprueba que el número de actualizaciónes no pueda ser igual o menor que 0.
Precondición	El videojuego debe estar iniciado. Debe existir el mapa "troncos".
Datos de entrada	actualizaciones = 0
Resultado esperado	Se lanza una excepción ValueError indicando que el parámetro "actualizaciones" debe ser mayor que 0.
Resultado obtenido	Se lanza una excepción ValueError indicando que el parámetro "actualizaciones" debe ser mayor que 0.

Tabla 4.26: Prueba de caja negra CN-07: Número de actualizaciones del entrenamiento.

CN-08	**Número de episodios que jugar**
Descripción	Comprueba que el número de episodios al jugar no pueda ser igual o menor que 0.
Precondición	El videojuego debe estar iniciado. Debe existir el mapa “troncos”.
Datos de entrada	episodios = 0
Resultado esperado	Se lanza una excepción ValueError indicando que el parámetro “episodios” debe ser mayor que 0.
Resultado obtenido	Se lanza una excepción ValueError indicando que el parámetro “episodios” debe ser mayor que 0.

Tabla 4.27: Prueba de caja negra CN-08: Número de episodios que jugar.

CN-09	**Colocar torres**
Descripción	Se coloca una torre en una posición válida y en una no válida para verificar que funciona correctamente.
Precondición	El videojuego debe estar iniciado. Debe existir el mapa “troncos”.
Datos de entrada	Para la posición válida: tipo de torre = 1, fila = 3, columna = 3; para la posición no válida: tipo de torre = 1, fila = 4, columna = 4
Resultado esperado	Se ha colocado únicamente un mono dardero en la posición (3, 3).
Resultado obtenido	Se ha colocado únicamente un mono dardero en la posición (3, 3).

Tabla 4.28: Prueba de caja negra CN-09: Colocar torres.

CN-10	**Generar mapa**
Descripción	Se comprueba que se genera la malla del mapa “troncos” y se almacena en el archivo “troncos.json”.
Precondición	El videojuego debe estar iniciado. No debe existir el fichero “troncos.json”.
Datos de entrada	mapa = “troncos”
Resultado esperado	Se ha generado el archivo “troncos.json” con una lista de listas de enteros.
Resultado obtenido	Se ha generado el archivo “troncos.json” con una lista de listas de enteros.

Tabla 4.29: Prueba de caja negra CN-10: Generar mapa.

CN-11	**Reiniciar partida**
Descripción	Comprueba si el agente es capaz de iniciar una nueva partida tras terminar un episodio
Precondición	El videojuego debe estar iniciado, y se ha entrado en el desafío donde se entrena al agente.
Datos de entrada	Ejecutar el método reiniciarPartida al ganar una partida, y otra vez al perderla.
Resultado esperado	Se ha iniciado una nueva partida en ambos casos.
Resultado obtenido	Se ha iniciado una nueva partida en ambos casos.

Tabla 4.30: Prueba de caja negra CN-11: Reiniciar partida.

Parte III

Resultados

Capítulo 5

Experimentación y evaluación

En este apartado se estudian diferentes implementaciones y cambios en el agente hasta obtener el producto final. Para ello, se establecen una serie de métricas con las que evaluar los resultados que generan los cambios y se comparan con el objetivo de mantener las características de mayor valor.

5.1. Diseño experimental

BloonsTD6 es un videojuego complejo: presenta numerosas variables que el agente inteligente debe aprender para alzarse con la victoria. Por ello, de igual forma que cualquier jugador comienza jugando en los niveles sencillos, el agente se enfrentará a desafíos sencillos que poco a poco aumentarán en dificultad. Para comenzar y comprobar las capacidades del agente inteligente, el primer reto que debe superar es completar las diez primeras rondas colocando un único tipo de torre, el "mono dardero". Tras esto, se aumentará la dificultad al restringir el número de torres máximas que podrá colocar. Finalmente, se elegirá otro mapa para evaluar cómo utiliza el agente lo aprendido.

5.1.1. Métricas

Las métricas son medidas cuantitativas del desempeño del agente durante el entrenamiento, permitiendo el seguimiento de su aprendizaje y la comparación con otras configuraciones de parámetros de aprendizaje y recompensas.

El módulo *stable-baselines3* facilita la generación de estas métricas a través de un registro de las mismas durante el entrenamiento, cada vez que se actualiza la política. Se reprensentan en gráficos (fig. 5.1) donde el eje de abcisas contiene el número de iteraciones (actualizaciones). Las métricas que resultan interesantes para evaluar la evolución del agente son las siguientes:

Recompensa media (ep_rew_mean) Recompensa final media de los episodios jugados.

Longitud media (ep_len_mean) Longitud promedia de los episodios jugados.

Pérdida (loss) Diferencia entre la recompensa esperada y la recompensa real obtenida. Durante la fase de exploración (fase inicial, el agente prueba diferentes conjuntos de acciones para comprender el entorno) este valor aumentará al encontrar nuevas formas de mejorar

la recompensa; al terminar esta fase, se mantendrá estable hasta comenzar a decaer, al acercarse el agente a la solución óptima.

Varianza explicada (explained_variance) Representa la capacidad del agente de predecir las recompensas que generarán las acciones. Un valor mayor o igual que uno significa que el agente es capaz de predecir la recompensas sin error, y un valor menor o igual a cero significa que el agente es incapaz de predecirlas. Según avance el entrenamiento, este valor debe ir aumentando paulatinamente hasta mantenerse estable y cercano a uno.

Entropía (entropy_loss) Simboliza la probabilidad que tienen las acciones de ser escogidas por el agente. Se define en el rango $(-\infty, 0]$, donde cero significa que el agente sólo escogerá una acción, y según se reduce el valor aumenta el conjunto de acciones que elegirá. Según avance el entrenamiento, este valor debe ir aumentando paulatinamente hasta mantenerse estable.

kl aproximado (approx_kl) Representa la diferencia de desempeño entre la política anterior y la nueva. Los picos representan una reducción drástica de rendimiento. Este valor debe disminuir paulatinamente durante el entrenamiento, hasta mantenerse estable y cercano al cero.

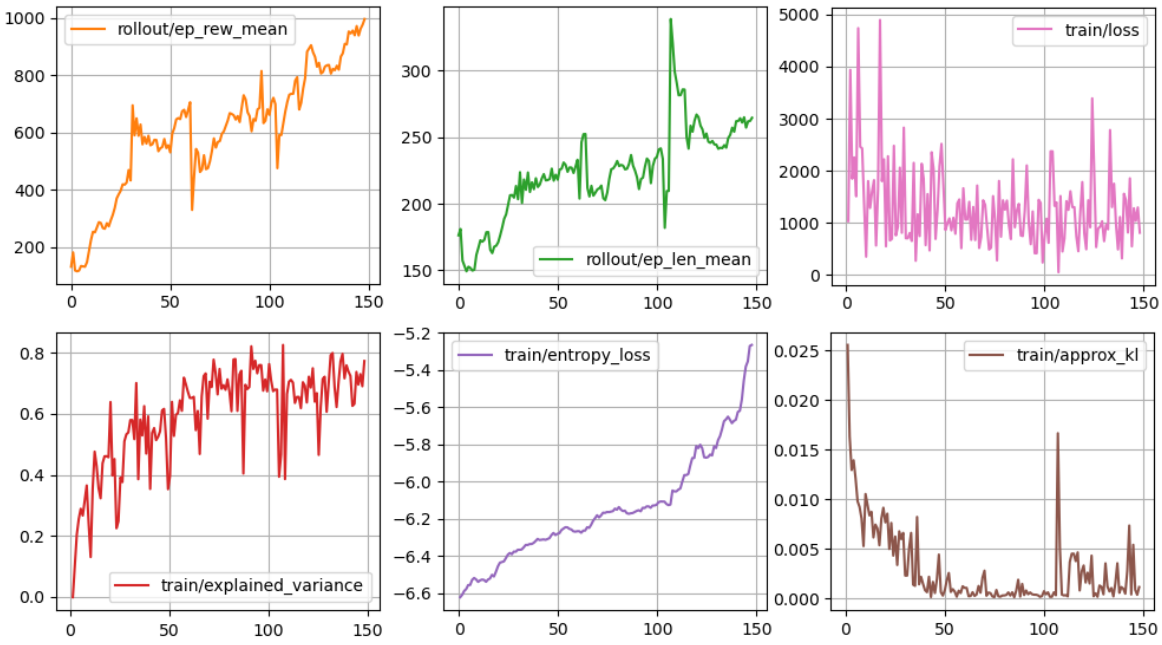


Figura 5.1: Diagramas de las métricas del entrenamiento

5.1.2. Proceso de evaluación

Una vez terminado el entrenamiento, el agente juega diez partidas (es el número de episodios que se utiliza para una actualización en el aprendizaje) y se calcula la media y desviación media

de las recompensas obtenidas, además de tener en cuenta el número de partidas que ha superado. Se considera que el agente es válido si supera al menos nueve partidas en el mapa en que ha entrenado con una media superior a 1000. También se le evaluará en otro mapa diferente, para estudiar si es capaz de utilizar lo aprendido en un entorno nuevo para el agente.

5.2. Experimentación y resultados

El primer reto al que se enfrentó el agente fue superar las diez primeras oleadas de enemigos el mapa "Troncos", pudiendo colocar únicamente monos darderos y comenzando cada partida con 650 monedas y 200 vidas (parámetros de una partida estándar en modo fácil). Esto resultó ser demasiado fácil, ya que el agente ganaba siempre, incluso colocando torres lejos del camino (fig. 5.2).

Figura 5.2: Estado de la partida al terminar el primer reto

Entonces, el siguiente paso fue añadir algunas limitaciones: por un lado, se redujo el dinero inicial a 170 monedas y las vidas a cinco; por otro, se limitó el número de torres a dos, el mínimo posible para superar este reto. Estos cambios provocan que sea necesario colocar las torres cerca del camino para poder ganar, además de reducir drásticamente el margen de error, al permitir que sólo se puedan escapar cuatro enemigos.

La configuración del aprendizaje consistía en 256 pasos por actualización y una constante de aprendizaje igual a 0.001. Se otorgaba una recompensa por cada vida que mantuviese al ganar la partida, con el objetivo de fomentar buenas posiciones de las torres que eviten la pérdida de

vidas. Tras entrenarlo por 80 iteraciones, los resultados no resultaron satisfactorios (fig. 5.3): la recompensa media era cercana a cero, y como la varianza explicada era menor que cero, el agente era incapaz de predecir la recompensa.

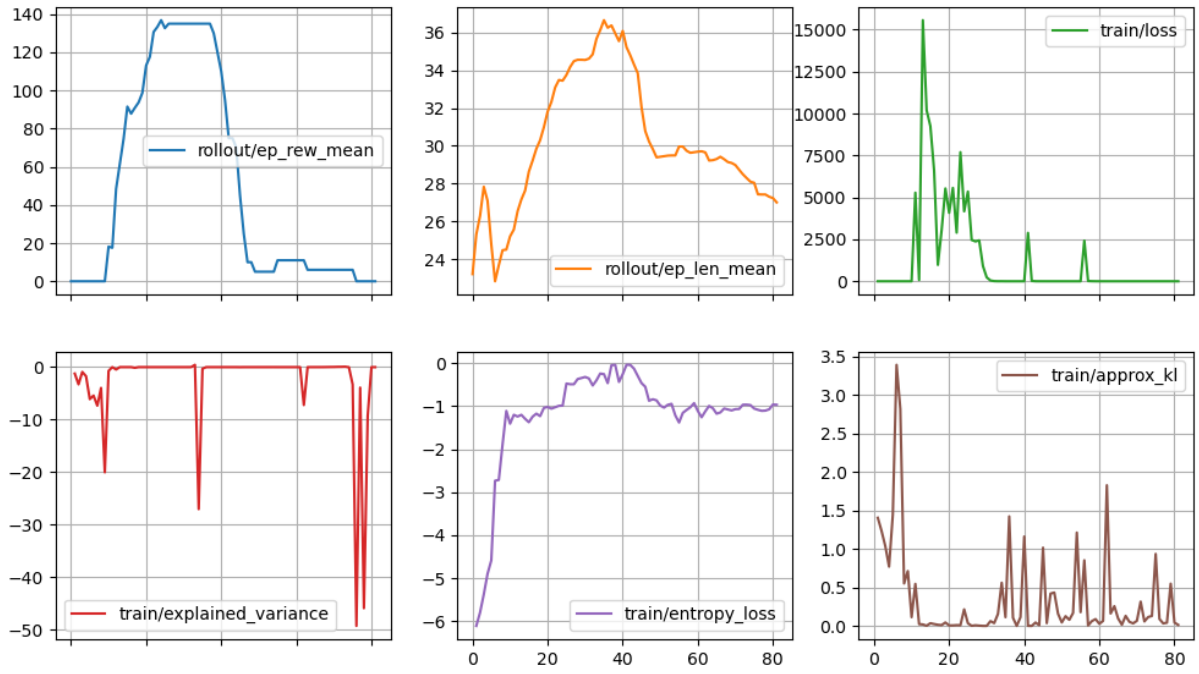


Figura 5.3: Métricas del primer entrenamiento

El problema era que el agente intentaba colocar la segunda torre encima de la primera, aunque no pudiese. La solución fue añadir un castigo cada vez que intentase colocar una torre en una posición no permitida. El problema ahora es que aprende que lo mejor es no colocar torres, porque si se equivoca pierde recompensa. Entonces, es necesario añadir otras recompensas y castigos más graduales. Se agregó una recompensa exponencial por ronda, para que sea considerablemente mayor en las últimas rondas. Junto a esto, se castigaba perder la partida y no haber colocado ninguna torre al perder. El resultado era mejor, las métricas (fig. 5.4) evolucionaban ligeramente como lo esperado, pero estaba lejos de ser el comportamiento deseado.

Tras investigar sobre los parámetros del aprendizaje [6], la constante de aprendizaje se fijó en 0.0003, se aumentó el número de pasos por actualización a 2048 pasos y se aumentó el número de actualizaciones por entrenamiento. Junto a esto, se redujo el tiempo de espera entre pasos de 0.2 segundos a 0.05, aumentando el número de pasos por partida. Además, se agregó un castigo por cada torre colocada con 0 reventones, ya que esto equivale a no haber colocado esa torre.

Tras entrenar al agente durante cuatro horas, los resultados fueron positivos (fig. 5.5): pese a que la recompensa media no crecía, la varianza explicada se mantuvo cercana al uno (el agente predice la recompensa correctamente), la entropía creació paulatinamente y el kl aproximado se redujo suavemente (aunque se mantuvo siempre cercano a cero, lo cual también es bueno). Este parecía el camino indicado.

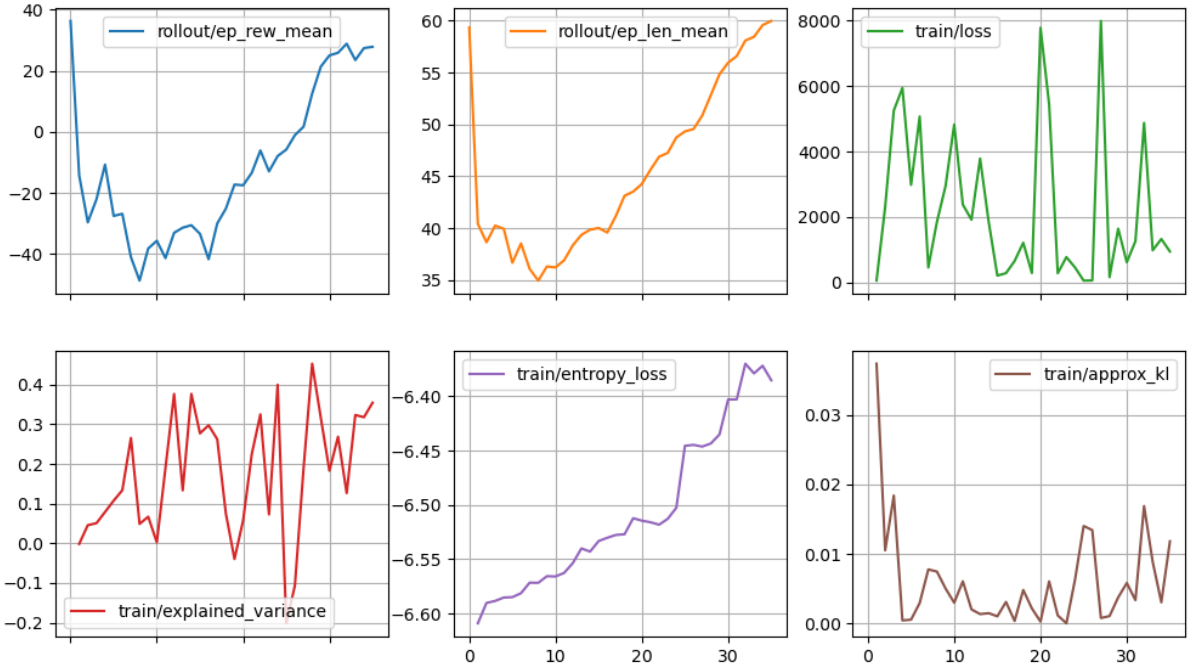


Figura 5.4: Métricas del segundo entrenamiento

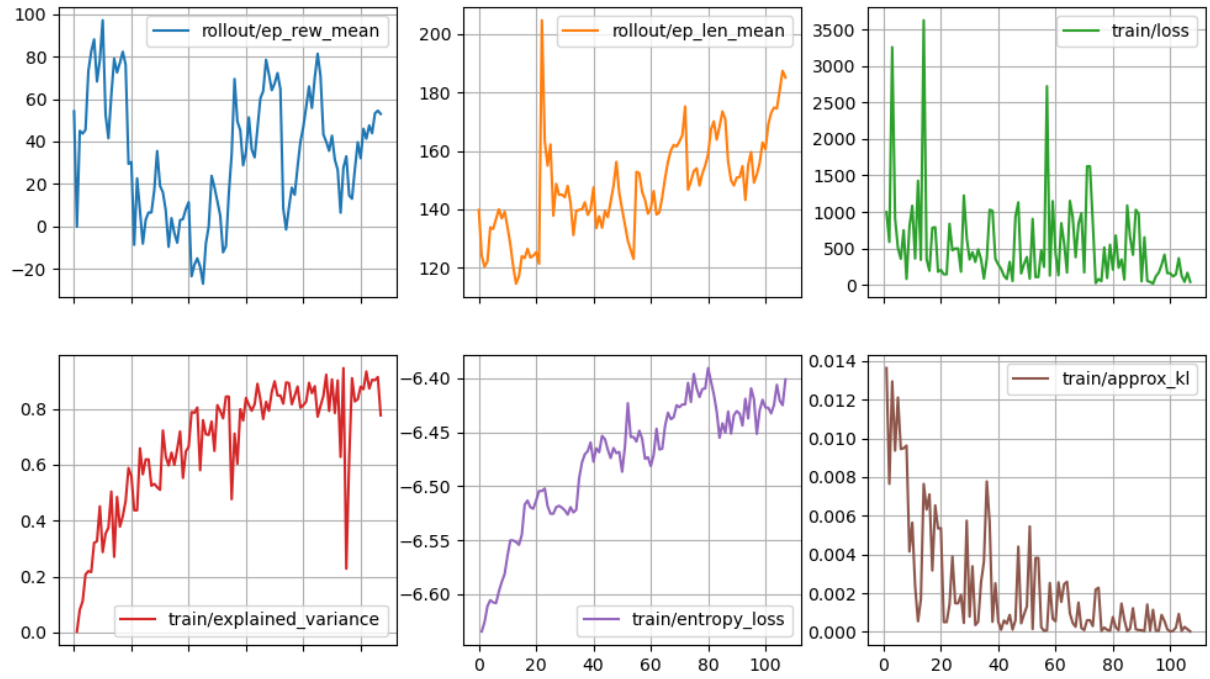


Figura 5.5: Métricas del tercer entrenamiento

Entonces, el último paso es conseguir que el agente alcance una recompensa media creciente, es decir, que tome mejores decisiones para ganar más partidas con todas las vidas. Esto se consigue modificando las recompensas. El cambio decisivo fue agregar recompensas por adyacencia: por cada camino adyacente a una torre, se recompensa al agente, y por cada torre, obstáculo o límite del mapa, se le castiga. Este cambio provoca que el agente coloque torres en lugares donde se crucen varios caminos, abarcando una buena zona del camino; además, castiga que se coloquen torres muy alejadas de los enemigos. Otro cambio a las recompensas incorporado es castigar por perder pudiendo haber colocado otro mono.

El resultado es el esperado (fig. 5.6): la recompensa crece con las iteraciones, la pérdida al inicio es más grande y se va reduciendo, la varianza explicada se mantiene cercana al 0.8 (predice razonablemente bien), la entropía crece constantemente y el kl aproximado permanece cercano al cero.

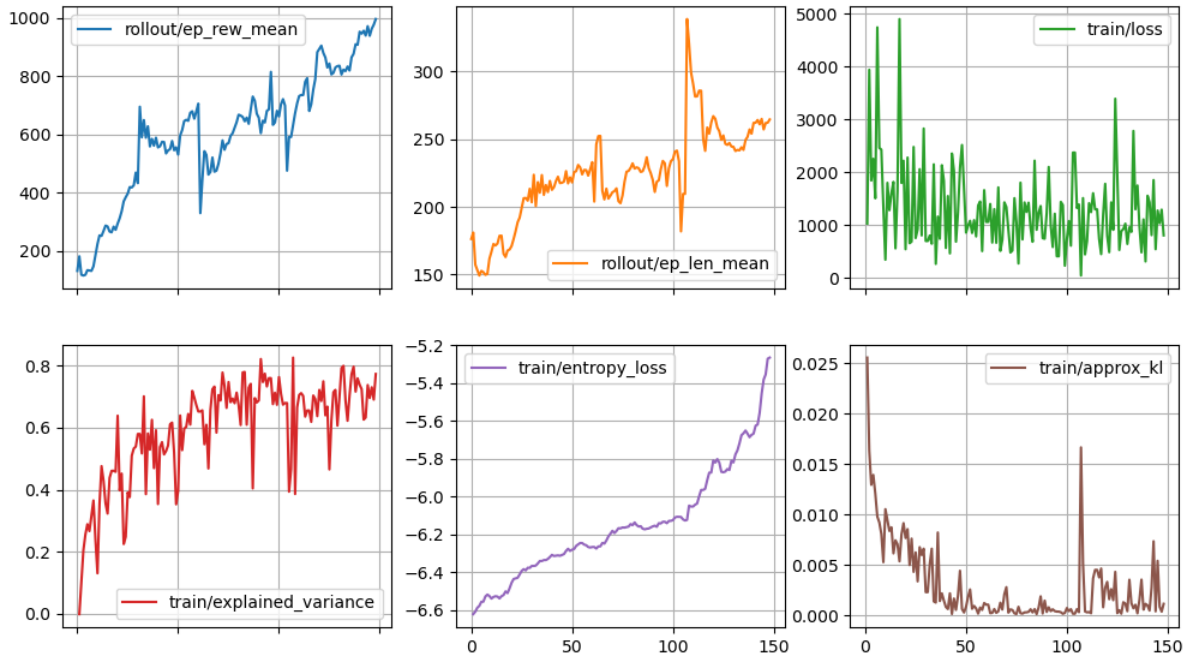


Figura 5.6: Métricas del cuarto entrenamiento

Entonces, se procedió a evaluarlo, primero en el mapa donde había estado entrenando ("troncos") donde jugó diez partidas, las cuales ganó todas con una recomensa media de 1011.33 y una varianza de 309.59. A continuación, jugó otras diez partidas en el mapa "cubismo" (fig. 5.7), que es más complicado, donde ganó 3 partidas con una media de 369.8 y una varianza de 533.08.

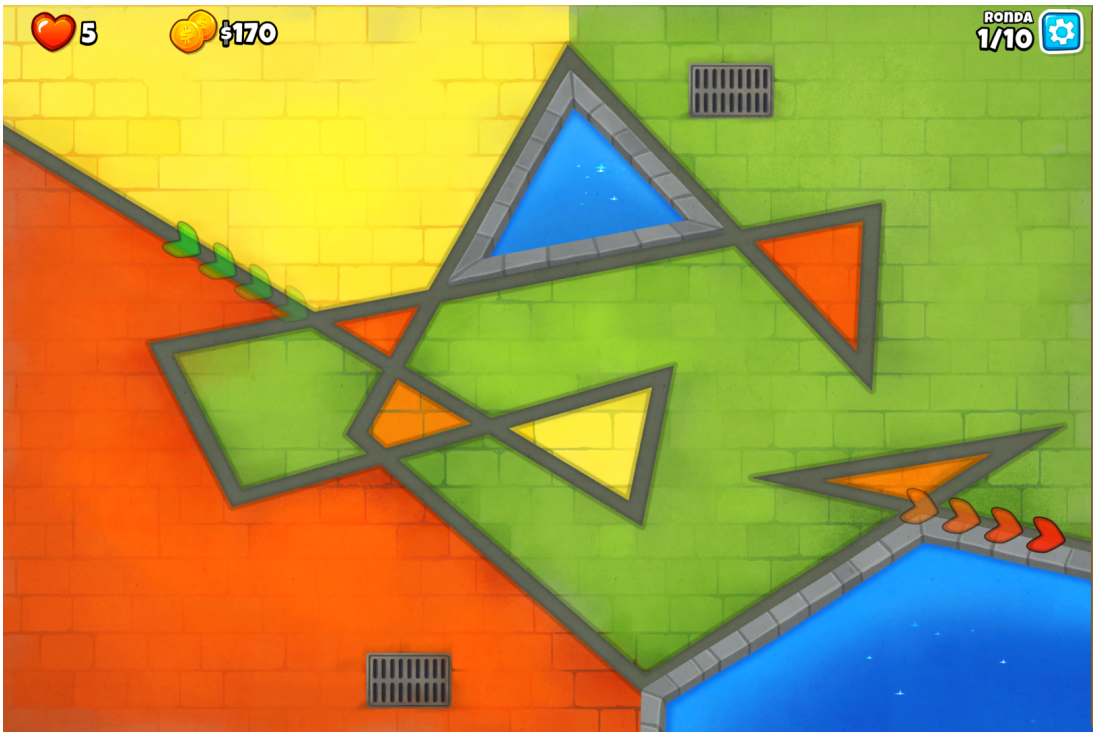


Figura 5.7: Mapa “cubismo”

5.3. Discusión de resultados

Los resultados de la evaluación del agente son buenos: es capaz de ganar de forma constante en el mapa en el que ha entrenado, y es capaz de encontrar posiciones estratégicas en otros mapas, como se ve en la figura 5.8, pero comete errores básicos en contadas ocasiones, como colocar torres alejadas del camino. En parte es de esperar que no tenga tan buen rendimiento en otros mapas, porque sólo ha entrenado en uno. Si se continuase entrenando en el mismo mapa, el agente se sobreajustaría, es decir, aprendería estrategias que resultasen útiles únicamente en "troncos", donde está entrenando. Si se alternase el aprendizaje entre dos mapas o más, el agente sería capaz de adaptarse a mapas nuevos más fácilmente, pero esto está fuera del alcance del proyecto.

Figura 5.8: Episodio de evaluación en el mapa "cubismo"

Capítulo 6

Conclusiones y trabajo futuro

En este apartado se recapitulan los principales aspectos del proyecto. Se interpretan los resultados obtenidos y el grado de cumplimiento con los objetivos, estudiando las limitaciones del proyecto y proponiendo posibles mejoras del sistema *software*. Junto a esto se incluye la perspectiva personal del estudiante que desarrolla este proyecto.

6.1. Conclusiones

El aprendizaje por refuerzo profundo es una herramienta muy potente, pero compleja. El desarrollo de este agente inteligente refleja el esfuerzo y recursos que se requieren para diseñar e implementar esta tecnología en un entorno complejo como es BloonsTD6. El agente es capaz de superar una versión simplificada del videojuego, lo cual es un buen avance, pero sólo es una pequeña parte de lo que puede llegar a alcanzar.

Realizar las estimaciones, el análisis y el diseño ha sido complejo, debido a la falta de experiencia en este campo. Ha sido necesario un gran esfuerzo de búsqueda de información al respecto, y ha sido beneficioso la existencia de un proyecto parecido (el agente de B2Studios, ver apartado 3.3) para hacerse una idea de qué se podía esperar en el desarrollo del agente.

Los cambios en la recompensa al final del proyecto han resultado ser una tarea tediosa, ya que es necesario entrenar al agente durante varias horas para poder ver las diferencias en los diagramas de las métricas. Esto ha supuesto estar atento del agente básicamente todos el día. Además, al sólo disponer de un ordenador, si el agente estaba entrenando no podía avanzar en otras áreas, por lo que la coordinación entre entrenamiento y desarollo ha sido un factor clave.

6.1.1. Perspectiva del proyecto

El agente desarrollado se ha ajustado perfectamente a los objetivos definidos (apartado 1.2). Se ha descrito el videojuego como un entorno con el que interactúa el agente, de forma escalable para poder agregar las características de BloonsTD6 que están fuera del alcance. Junto a esto se ha desarrollado un agente con unos sensores y actuadores robustos y eficientes, en especial los sensores con la extensión Puente, que permite cambios en los parámetros que definen el estado de forma más sencilla y flexible. Finalmente, el agente ha estado entrenando lo suficiente como para superar el mapa "troncos" de forma consistente e, incluso, otros mapas que nunca ha probado.

Pese a que el agente en su estado actual tiene un conocimiento muy básico del juego, tras la inclusión de más características del videojuego será capaz de superar los diferentes modos de

juego en un rango mayor de mapas. Esto permitirá, por un lado, descubrir nuevas estrategias y formas de utilizar a las torres en conjunto, lo cual resulta útil para evaluar el desempeño de las torres y facilitar el cambio de sus atributos en las actualizaciones. Por otro lado, este agente resultará útil para el modo cooperativo, donde un jugador que no quiera jugar a través de Internet podrá tener como compañero al agente.

La metodología ASAP ha resultado realmente útil a la hora de gestionar el proyecto, especialmente por el contacto frecuente con los tutores, lo cual ha permitido mantener un ritmo de trabajo estable y constante, detectar bloqueos que no parecían existir y encontrar soluciones adecuadas en el menor tiempo posible. En especial, la comunicación de progresos ha resultado un elemento clave en el proyecto, porque además de practicar y refinar la presentación del TFG, el debate tras las presentaciones es muy beneficioso al recibir otros puntos de vista de los miembros de la comunidad.

La división en *sprints* ha facilitado la organización del proyecto, y han permitido centrarse en un conjunto de objetivos, que resulta más sencillo que enfrentarse al proyecto completo. Aunque en la reunión de inicio se definen las tareas que se llevarán a cabo durante el *sprint*, al no conocer exactamente qué tareas eran necesarias por falta de experiencia en las tecnologías, se iban dividiendo las tareas más grandes en tareas más pequeñas mientras se iban completando, decidiendo en las reuniones de sincronización qué tareas se llevarían a cabo.

6.1.2. Perspectiva personal

Al iniciar el proyecto no conocía prácticamente nada del aprendizaje por refuerzo profundo, por lo que no sabía realmente qué objetivos serían realistas. En el primer *sprint* definí un alcance exageradamente amplio, ya que quería desarrollar un agente que superase el modo de dificultad más difícil, "CHIMPS", pero para ello el agente tiene que ser capaz de mejorar a los monos y tener acceso a un mayor abanico de torres, por lo que en el segundo *sprint* reduje considerablemente el alcance siguiendo el *feedback* de los tutores.

Ha sido muy satisfactorio aprender sobre este campo y ver al agente jugar de forma autónoma. Me resulta fascinante el concepto de que un programa sea capaz de aprender de forma similar a los humanos. He aprendido mucho más de lo que esperaba sobre el aprendizaje automático gracias a este proyecto. Me gustaría haber desarrollado otros componentes del agente, como las mejoras o algún otro tipo de torre más, pero los resultados obtenidos son igualmente satisfactorios.

Considero que ha sigo un factor clave el uso de ASAP como metodología del proyecto, ya que ha hecho más intuitiva la gestión y planificación del proyecto. El factor que más me ha gustado de esta metodología es el impulso que da a la comunicación del equipo, que en lo personal me ayuda a evitar el "efecto túnel", junto con otros aspectos, como la práctica de la defensa del TFG junto con el posterior debate que otorga nuevos puntos de vista, tanto por las opiniones de otros compañeros como por ver el desarrollo de otros TFGs relacionados.

Aunque algunos aspectos de la memoria me han resultado más cargantes o complejos, desarrollar este proyecto me ha resultado muy satisfactorio e interesante, en especial el diseño y programación de los sensores y actuadores, mientras que en algunos momentos me he sentido un poco perdido al diseñar el modelo del agente y las recompensas (aunque he de reconocer que me puse un poco nervioso al ser incapaz de recuperar los punteros tras la actualización).

6.2. Trabajo futuro

La versión actual del agente es capaz de utilizar muy pocos aspectos de BloonsTD6. El siguiente paso a dar sería aumentar el abanico de torres para incluir a los monos ninja y los cañones, para poder reventar Bloons camuflados y de plomo, y de esta forma poder entrenar al agente para que supere el modo de dificultad "fácil", es decir, hasta la ronda 40.

Tras esto, seguiría añadir la capacidad de mejorar a las torres, un factor clave para poder enfrentarse a dificultades mayores. En BloonsTD6 suele ser mejor tener pocas torres poderosas que tener muchas débiles, ya que las mejoras traen consigo características únicas para las torres. Pudiendo mejorar monos, el agente sería capaz de superar el modo de dificultad "intermedio" e, incluso, "difícil".

Si se aumenta una vez más el abanico de monos para incluir torres de apoyo (ralentizar globos, mejoras temporales a las torres, etc) y algunas otras torres de ataque más, el agente se podría enfrentar al modo de dificultad "CHIMPS", consiguiendo de esta forma superar los modos de dificultad estándar, ya que los otros modos son variaciones de los mismos. Si tras esto se realiza el entrenamiento cambiando el mapa tras varios episodios, el agente sería capaz de superar estos modos en más mapas, dando como resultado un agente muy completo.

El problema es que al aumentar el número de variables (más torres, mejoras, mapas, partidas más largas, etc), el entrenamiento será cada vez más complejo y requerirá más tiempo y recursos. No sólo eso, si no que hay que definir las recompensas para guiar su comportamiento, que es la tarea más complicada de diseñar al agente.

Parte IV

Apéndices

Apéndice A

Manual de Instalación

Para poder utilizar a este agente inteligente, se necesita una copia del videojuego BloonsTD6 y Python (o Anaconda) instalados. En este manual se utiliza una copia obtenida de la plataforma Steam [7], por lo que la instalación de extensiones puede diferir en versiones procedentes de otras plataformas.

A.1. Añadir las extensiones a BloonsTD6

Lo primero es descargar MelonLoader desde este enlace: [14]. También será necesario instalar .NET6, que se puede descargar desde este mismo enlace.

Una vez descargados, hay que realizar una copia del videojuego (para mantener el original). Para ello, hay que navegar hasta a `C:\ProgramFiles(x86)\Steam\steamapps\common` y copiar la carpeta "BloonsTD6" (fig. A.1). Esta será la copia que se utilizará para el agente, y a la que se refieren los siguientes pasos. Tras esto, descomprimir el archivo `MelonLoader.x64.zip` en la carpeta del juego (fig. A.2).

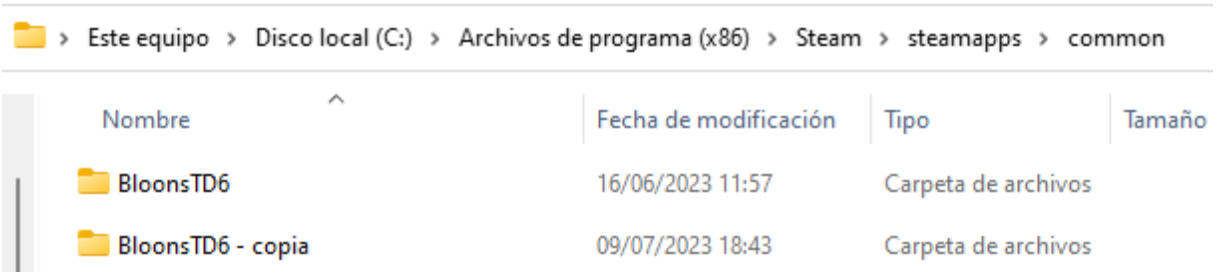


Figura A.1: Ruta de BloonsTD6

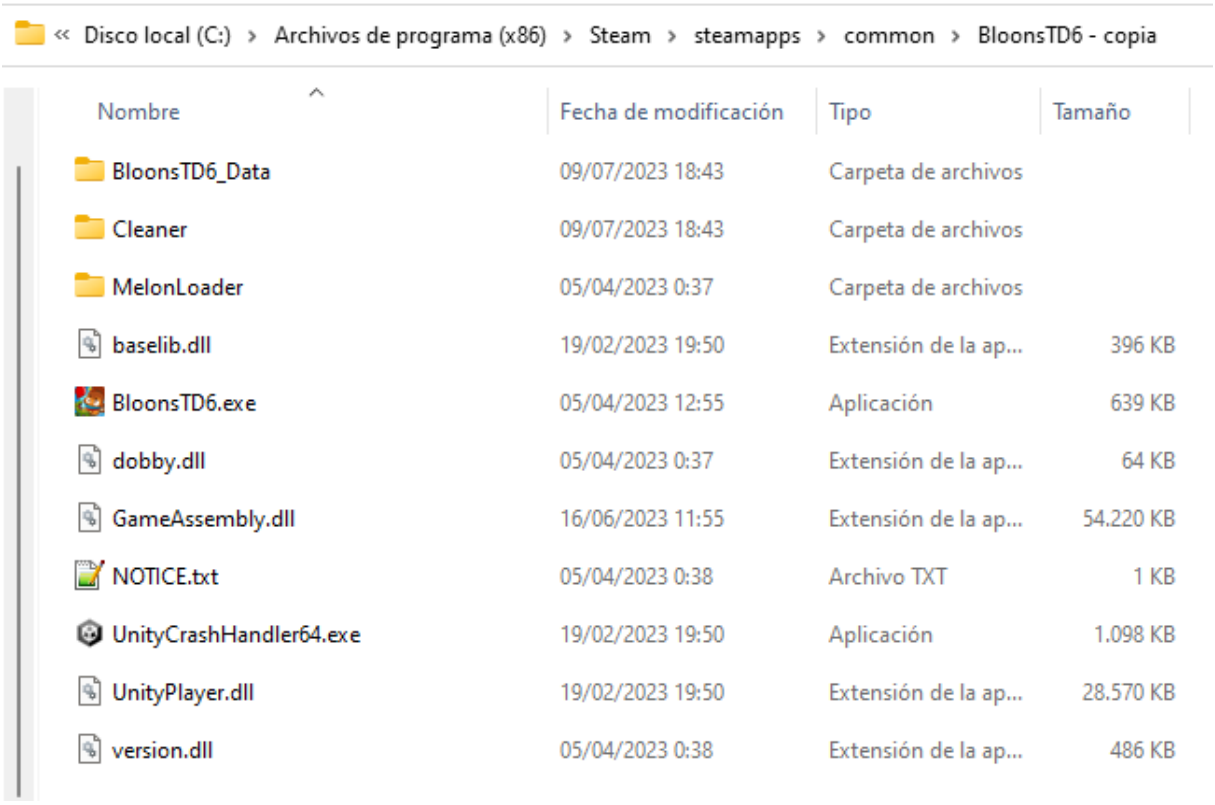


Figura A.2: Ruta de BloonsTD6 tras añadir MelonLoader

Con Steam abierto en segundo plano, iniciar el juego ejecutando`BloonsTD6.exe`, para que MelonLoader genere los archivos y carpetas necesarios. Cuando aparezca la pantalla principal, cerrar el juego. A continuación, se debe descargar `Btd6ModHelper.dll` desde su página de proyecto en GitHub [37]. Una vez descargado, colocar el fichero `Btd6ModHelper.dll` en la carpeta `BloonsTD6-copia\Mods`.

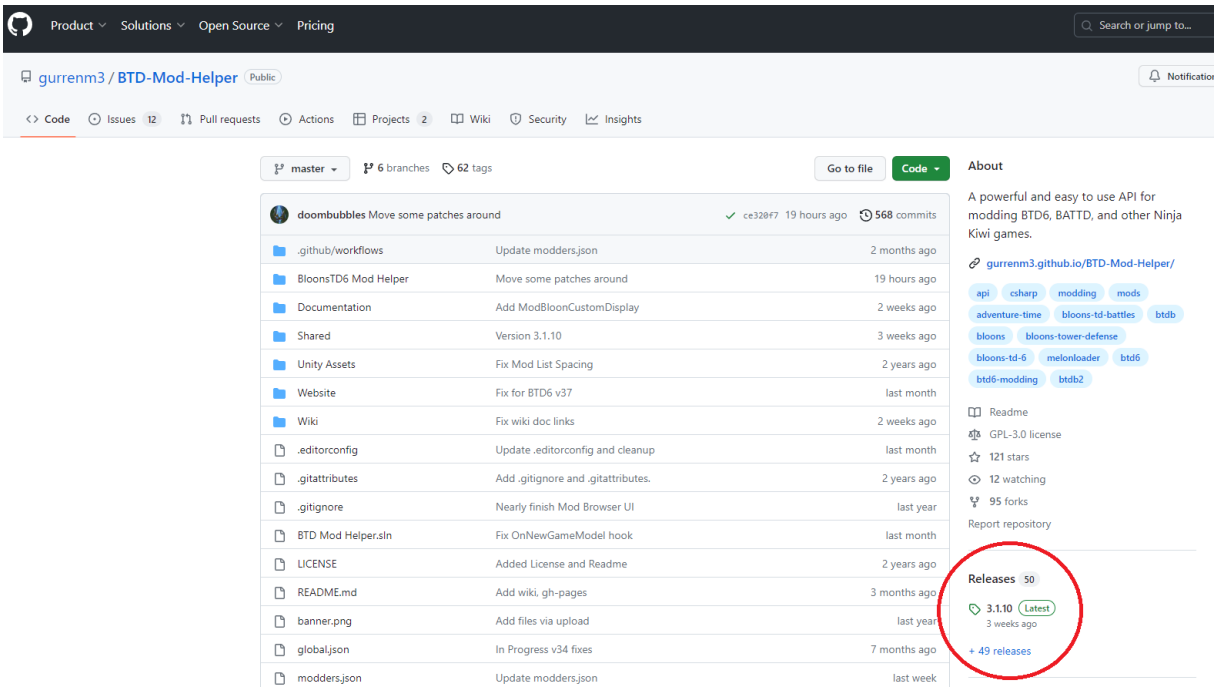
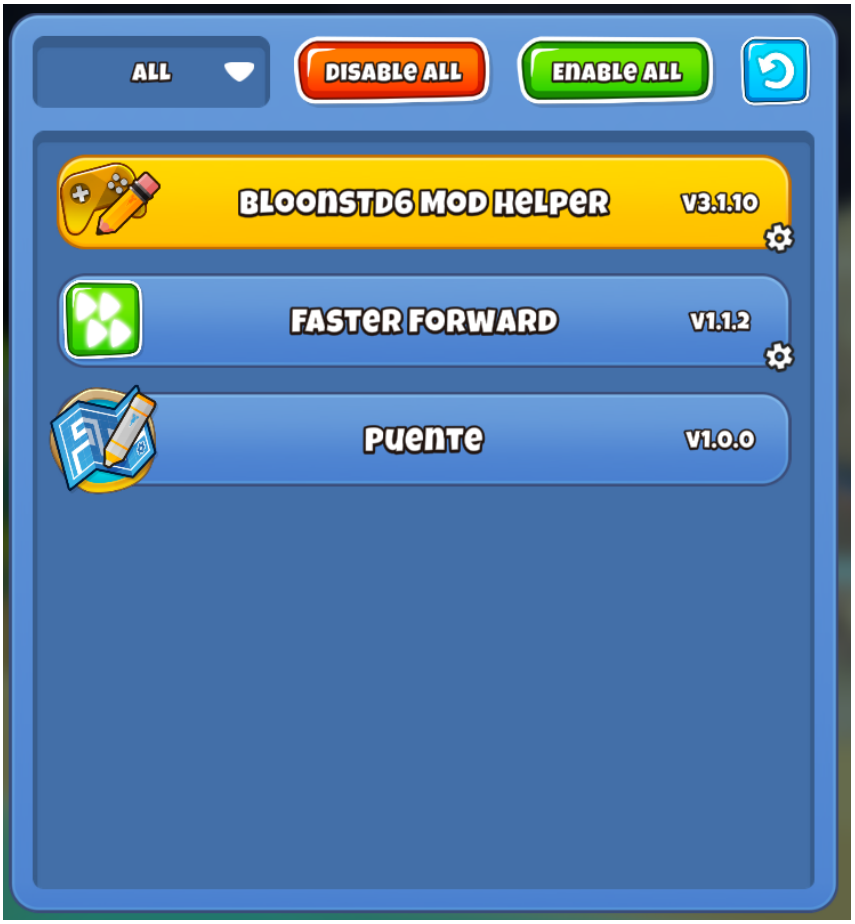


Figura A.3: Página de GitHub de Btd6ModHelper

Iniciar otra vez el juego e ir al menú principal. Si aparece en la esquina inferior derecha aparece el icono de la figura A.4a, la extensión `Btd6ModHelper.dll` se ha instalado correctamente. El siguiente paso es descargar el archivo `FasterForward.dll` [16] y colocarlo en la carpeta `BloonsTD6-copia\Mods`, y de igual forma colocar en esa carpeta la extensión Puente, `Puente.dll`. Iniciar el juego una vez más, ir al menú pricipal y clicar el icono de la figura A.4a. Si aparecen las tres extensiones listadas (fig. A.4b, el videojuego está preparado.

(a) Icono "Mods"



(b) Lista de extensiones instaladas

Figura A.4: Resultado de la instalación de las extensiones

A.2. Instalación de librerías

Las librerías necesarias para la ejecución del agente están recogidas en el archivo `librerias.txt`. Para instalarlas, simplemente hay que abrir un terminal (si se utiliza Anaconda, abrir un terminal del entorno) y ejecutar el comando `pip install -r` "ruta/a/librerias.txt".

Apéndice B

Manual de Usuario

B.1. Módulo Agente

El módulo principal es `Agente` que, como su nombre indica, representa al agente inteligente. El primer paso es instanciar a un Agente, que tiene los siguiente parámetros:

`mapa : str` Nombre del mapa donde jugará el agente. Hay disponibles dos mapas, "troncos" y "cubismo", pero se pueden generar más con el módulo `Mapas`.

`maxTorres : int` Número máximo de torres que se pueden colocar en una partida.

`rutaModelo = ''` Archivo .zip de un modelo con el que inicializar al agente.

`tasaAprendizaje = 0.001` Tasa de aprendizaje para el entrenamiento.

`pasosPorActualizacion = 2048` Número de pasos que se utilizarán para actualizar la política del agente.

El método `entrenar(actualizaciones : int)` permite entrenar al agente el número de actualizaciones indicado. El proceso de entrenamiento genera el archivo `ppo_btd6.zip`, que contiene al modelo entrenado. Además, en la carpeta `entrenamientos/"dia_actual"/"hora_actual"/` se encuentran las copias de seguridad del modelo (una por actualización) y los registros del entrenamiento (para ver con el módulo `Diagramas`. En el siguiente ejemplo se entrena un nuevo agente en el mapa "troncos" con un número máximo de 2 torres durante 32 actualizaciones (en torno a una hora).

```
from Agente import Agente

agente = Agente('troncos', 2)
Agente.entrenar(32)
```

El método `jugar(episodios : int)` permite al agente jugar durante el número de episodios indicado, devolviendo la media y desviación típica de la recompensa obtenida en cada partida. En el siguiente ejemplo el agente juega diez partidas en el mapa "cubismo" con un máximo de tres torres, y con el modelo `ppo_btd6.zip`.

```
from Agente import Agente

agente = Agente('cubismo', 3, 'ppo_btd6.zip')
Agente.jugar(10)
```

B.2. Módulo Diagramas

Este módulo contiene la función `mostrarDiagramas(ruta : str, archivos = 1`, que genera una imagen en formato .png con los diagramas de las métricas de un entrenamiento. Hay dos formas de utilizarlo: si el entrenamiento sólo ha tenido una sesión, se pasa como parámetro la ruta de la carpeta del entrenamiento (`entrenamientos/"dia_actual"/"hora_actual"/`). Ejemplo: generar los diagramas de un entrenamiento del día 20/06/2023 a las 14:13:

```
from Diagramas import mostrarDiagramas

mostrarDiagramas('entrenamientos/20230620/14_13')
```

Si el entrenamiento ha tenido varias sesiones, se deben guardar en una carpeta los directorios de los diferentes entrenamientos con el número de sesión que representa. Ejemplo: La carpeta `adyacencia` contiene seis sesiones de entrenamiento cuyas carpetas están numeradas del uno al seis:

```
entrenamientos
|-- adyacencia
    |-- 1
    |-- 2
    |-- 3
    |-- 4
    |-- 5
    |-- 6

from Diagramas import mostrarDiagramas

mostrarDiagramas('entrenamientos/adyacencia/', 6)
```

B.3. Módulo Mapas

Este módulo contiene la clase Mapas con varios métodos estáticos, siendo los más relevantes existeMapa(mapa : str), que devuelve True si existe el fichero JSON del mapa en la carpeta Mapas, y False en caso contrario; y generarMapa(mapa : str), que genera el fichero JSON del mapa en la carpeta mapas. Para generar un mapa, hay que entrar en el mapa deseado en el juego en el modo "fácil: entorno de pruebas", y ejecutar la función generarMapa. Cuando se hayan colocado todas las torres posibles (fig. B.1), se habrá generado el fichero JSON.

Figura B.1: Generación del mapa

Bibliografía

[1] *Agents in Artificial Intelligence*. GeeksforGeeks. Section: Advanced Data Structure. 28 de ene. de 2018. URL: https://www.geeksforgeeks.org/agents-artificial-intelligence/ (visitado 06-07-2023).

[2] *AI in Gaming (Artificial Intelligence is Coming)*. Section: Gaming. 1 de mar. de 2022. URL: https://www.gamedesigning.org/gaming/ai-in-gaming/ (visitado 04-07-2023).

[3] *AlphaStar: Mastering the real-time strategy game StarCraft II*. URL: https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii (visitado 30-03-2023).

[4] *Anaconda - The World's Most Popular Data Science Platform*. URL: https://www.anaconda.com/ (visitado 05-07-2023).

[5] b2studios. *Beating BTD6 with AI*. 20 de ago. de 2021. URL: https://www.youtube.com/watch?v=QoEMoSbGvbM (visitado 30-03-2023).

[6] *best-practices-ppo.md*. GitHub. URL: https://github.com/llSourcell/Unity_ML_Agents (visitado 02-07-2023).

[7] *Bienvenidos a Steam*. URL: https://store.steampowered.com/ (visitado 09-07-2023).

[8] P. Bourque y R.E. Fairley. *Guide to the Software Engineering Body of Knowledge, Version 3.0*. ACM-IEEE. 2014.

[9] *Búsqueda de empleo en Glassdoor*. Glassdoor. URL: https://www.glassdoor.es/index.htm (visitado 12-07-2023).

[10] *ChatGPT*. URL: https://openai.com/chatgpt (visitado 04-07-2023).

[11] *Cheat Engine*. URL: https://www.cheatengine.org/ (visitado 03-04-2023).

[12] This text provides general information Statista assumes no liability for the information given being complete or correct Due to varying update cycles y Statistics Can Display More up-to-Date Data Than Referenced in the Text. *Topic: Video game industry*. Statista. URL: https://www.statista.com/topics/868/video-games/ (visitado 03-07-2023).

[13] *Difficulty*. Bloons Wiki. 6 de jul. de 2023. URL: https://bloons.fandom.com/wiki/Difficulty (visitado 06-07-2023).

[14] *Downloads and Installation*. btd6-modding-tutorial. URL: https://hemisemidemipresent.github.io/btd6-modding-tutorial/ (visitado 05-07-2023).

[15] *Function Point Languages Table*. Quantitative Software Management. Agosto 2015. URL: http://www.qsm.com/resources/function-point-languages-table.

[16] James Gale. *Faster Forward*. original-date: 2022-04-24T19:47:58Z. 2 de jul. de 2023. URL: https://github.com/doombubbles/faster-forward (visitado 05-07-2023).

[17] *Gestiona los proyectos de tu equipo desde cualquier lugar | Trello*. URL: https://trello.com/es (visitado 10-04-2023).

[18] *GitHub - Farama-Foundation/Gymnasium: An API standard for single-agent reinforcement learning environments, with popular reference environments and related utilities (formerly Gym)*. URL: https://github.com/Farama-Foundation/Gymnasium (visitado 05-07-2023).

[19] *History of AI Use in Video Game Design*. Big Data Analytics News. 24 de mar. de 2021. URL: https://bigdataanalyticsnews.com/history-of-artificial-intelligence-in-video-games/ (visitado 07-07-2023).

[20] Jasurbek. *Atari Game*. Medium. 7 de jun. de 2023. URL: https://medium.com/@jaskabratan/atari-game-ef418fdeb0e7 (visitado 06-07-2023).

[21] Ben Johnson. *PyDirectInput*. original-date: 2020-02-18T19:55:16Z. 31 de mar. de 2023. URL: https://github.com/learncodebygaming/pydirectinput (visitado 03-04-2023).

[22] *Keras*. URL: https://keras.io/ (visitado 03-04-2023).

[23] Ninja Kiwi. *Bloons TD 6 - Ninja Kiwi*. URL: https://ninjakiwi.com/Games/Mobile/Bloons-TD-6.html (visitado 31-03-2023).

[24] Yunlong Lu y Wenxin Li. «Techniques and Paradigms in Modern Game AI Systems». En: *Algorithms* 15.8 (2022). ISSN: 1999-4893. DOI: 10.3390/a15080282. URL: https://www.mdpi.com/1999-4893/15/8/282.

[25] Miguel A. Martínez-Prieto et al. «Una metodología basada en prácticas ágiles para la realización de Trabajos Fin de Grado». En: *Actas de las JENUI* 8 (2023).

[26] *Microsoft Teams*. URL: https://www.microsoft.com/es-es/microsoft-teams/free (visitado 05-07-2023).

[27] *Numpy*. original-date: 2010-09-13T23:02:39Z. 3 de abr. de 2023. URL: https://github.com/numpy/numpy (visitado 03-04-2023).

[28] *OpenAI Five defeats Dota 2 world champions*. URL: https://openai.com/research/openai-five-defeats-dota-2-world-champions (visitado 30-03-2023).

[29] *Overleaf*. URL: https://es.overleaf.com (visitado 05-07-2023).

[30] Wouter van Heeswijk PhD. *Proximal Policy Optimization (PPO) Explained*. Medium. 31 de ene. de 2023. URL: https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b (visitado 06-07-2023).

[31] R. Pressman. *Ingeniería del Software*. McGraw-Hill, 2014.

[32] SethBling. *MarI/O - Machine Learning for Video Games*. 13 de jun. de 2015. URL: https://www.youtube.com/watch?v=qv6UVOQ0F44 (visitado 30-03-2023).

[33] Adi Shamir. «How to share a secret». En: *Communications of the ACM* 22 (1979).

[34] srounet. *Pymem*. original-date: 2010-09-23T20:34:00Z. 3 de abr. de 2023. URL: https://github.com/srounet/Pymem (visitado 03-04-2023).

[35] *Stable Baselines3*. original-date: 2020-05-05T05:52:26Z. 3 de abr. de 2023. URL: https://github.com/DLR-RM/stable-baselines3 (visitado 03-04-2023).

[36] *Stable Diffusion Public Release*. Stability AI. URL: https://stability.ai/blog/stable-diffusion-public-release (visitado 04-07-2023).

[37] Thomas. *BTD Mod Helper*. original-date: 2021-03-12T03:34:46Z. 4 de jul. de 2023. URL: https://github.com/gurrenm3/BTD-Mod-Helper (visitado 05-07-2023).

[38] Two Minute Papers. *¡DeepMind Creó una IA que juega de forma sobrehumana 57 juegos de Atari!* 23 de mayo de 2020. URL: https://www.youtube.com/watch?v=dJ4rWhpAGFI (visitado 30-03-2023).

[39] *Términos y Condiciones de Ninja Kiwi*. URL: https://ninjakiwi.com/terms (visitado 03-04-2023).

[40] *Visual Studio 2022*. Visual Studio. URL: https://visualstudio.microsoft.com/es/vs/ (visitado 05-07-2023).

[41] *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (visitado 05-07-2023).